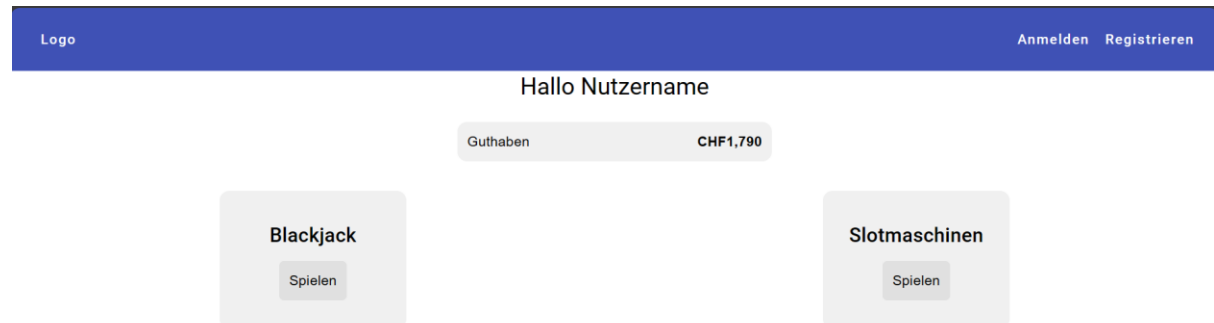


Casino App

Fullstack Angular Applikation



Individuelle praktische Arbeit

Paul Odiana Pietzko

Kantonsschule Sargans

Exp. Christian Steiner

05.03.2024

Inhalt

Inhalt	1
Kurzfassung	5
Ausgangssituation	5
Umsetzung	5
Ergebnis.....	5
Projektaufbauorganisation	6
Zeitplan	7
Arbeitsjournal	8
Tag 1 – 20.02.2024	8
Tag 2 – 27.02.2024.....	10
Tag 3 – 05.03.2024.....	11
Tag 4 – 12.03.2024.....	12
Tag 5 – 14.03.2024.....	13
Tag 6 – 19.03.2024.....	14
Tag 7 – 26.03.2024.....	15
Tag 8 – 02.04.2024.....	16
Tag 9 – 23.04.2024.....	17
Tag 10 – 30.04.2024.....	18
Tag 11 – 23.04.2024.....	19
Tag 12 – 05.05.2024.....	20
Tag 13 – 07.05.2024.....	21
Projektmanagementmethode	22
1 Informationsbeschaffung.....	23
1.1 Erwartbarkeit	23
1.2 Ziel.....	23
1.3 Unklarheiten	23
1.4 Projektumfeld	23
1.4.1 Abgrenzung	23
1.4.2 Umsysteme	24
1.4.3 Assets	24
1.4.4 Visual Studio Code	24
1.4.5 GitHub	24
1.4.5 MongoDB	24
1.4.6 Postman	24

1.4.7 Konventionen	24
1.5 Besprechung	25
2 Planung	26
2.1 Zeitplan	26
2.1.1 Rundung	26
2.1.2 Interpretation.....	26
2.1.3 Blocker	26
2.1.4 Ist-Zeit	26
2.1.5 Meilensteine	26
2.2 Testkonzept.....	27
2.2.1 Testmethoden.....	27
2.2.2 Testdaten	28
2.2.3 Testprotokoll	28
2.3 Testfälle	29
2.5 Datensicherung und Versionierung	30
2.6 Rest API	30
2.6.1 JWT.....	30
2.6.2 Fehlerbehebung	30
2.6.3 Mock Daten.....	30
2.7 Daten.....	31
2.7.1 Public Ordner	31
MongoDB	31
Sicherheit	31
2.8 Architektur	31
2.8.1 Frontend.....	32
2.9 Node Version.....	32
2.10	32
3 Entscheidung.....	33
3.1 Angular Setup.....	33
3.2 Packages.....	33
3.3 Daten.....	33
3.5 Sonstige Konventionen	33
4 Realisierung.....	34
4.1 Erstellung Git Repository	34
4.2 Angular Setup.....	34
4.2.1 Node Version auswählen.	34

4.2.2 Projekt Generieren.....	34
4.2.3 Pakete installieren	34
4.2.4 Aufräumen	35
4.3 Linting und Testing.....	35
4.3.1 Prettier	35
4.5 CI/CD	36
4.6 Branches.....	36
4.7 Backend.....	36
4.7.1 Server.JS	36
4.7.2 App.JS	36
4.7.3 AppError	36
4.7.4 Routes	36
4.7.5 User Schema	37
4.7.6 Controllers.....	38
4.8 Frontend.....	43
4.8.1 App.component.html.....	43
4.8.2 App Routes.....	44
4.8.3 Components.....	44
4.8.4 Game Logik.....	46
5 Kontrolle.....	56
5.1 Refactoring.....	56
5.1.1 Code Strukturierung.....	56
5.1.2 Clean Code	56
5.2 Testdurchführung	56
5.2.1 Basisarbeiten.....	57
5.3 Testauswertung.....	61
5.4 Verbesserungen	61
6 Auswertung.....	62
6.1 Vorgehen.....	62
6.1.1 Planung	62
6.2 Probleme.....	62
6.2.1 IPERKA.....	62
6.2 Zeitmanagement.....	62
6.3 Einhaltung Meilensteine	62
6.4 Resultat	63
6.5 Zukunft	64

Schlusswort	65
Glossar	66
Abbildungsverzeichnis	67
Quellenverzeichnis	68

Kurzfassung

Ausgangssituation

Im Umfeld einer Probe-IPA-Arbeit, bekommt die Gesamte Klasse 3A Zeit während eines Semesters eine Arbeit zu programmieren. Dabei ist der Kandidat in Eigenverantwortung seine Arbeit so zu wählen, dass er sie in der Vorgegebenen Zeit vollenden kann. Um die Stärken des Angular Frameworks austesten zu können, wurde im Rahmen dieser Arbeit eine spielerische Multiuser Webapplikation erstellt. In dieser können sich Nutzer registrieren und zwischen den beiden Spielmodi: BlackJack und Slot Maschinen zu entscheiden. Wie bei einer Echten online Casino App, wird auch hier der Kontostand über alle Spiele hinweg synchronisiert.

Umsetzung

Umgesetzt wurde das Projekt mithilfe von der Projektmanagementmethode IPERKA. Programmiert wurde mit Visual Studio Code als Entwicklungsumgebung und GitHub zur Datensicherung und Versionierung bzw. Protokollierung. Die Definierung von Meilensteinen sorgte für eine gute Übersicht der zu implementierenden Funktionen. Bei der Umsetzung dieser Webapplikation lag der Fokus auf dessen Funktionalität und der Umsetzung allgegenwärtigen Programmierkonventionen.

Folgendes wurde umgesetzt:

- **BlackJack:** Voll funktionstüchtiger BlackJack game loop mit Dealer KI.
- **SlotMaschine:** Spielautomat mit Spin Funktion.
- **API:** Eigen aufgesetztes Backend mit allen Systemwichtigen Endpoints.

Ergebnis

Das Ergebnis ist eine attraktive Webapplikation, welche von mehreren Nutzern gleichzeitig genutzt werden kann. Nach der Erstellung eines Kontos kann man mit dem Startkapital von 1'000 Franken in den Spielmodi BlackJack und Slot Maschinen sein Glück auf den Grossgewinn versuchen.

Der Code ist sauber und befolgt allgegenwärtigen Programmierkonventionen. So wird die Logik in einzelnen Dateien in Services heruntergebrochen.

Der Gesamte Code wird über GitHub versioniert bzw. protokolliert.

Teil 1

Dieser erste Teil enthält die Projektaufbauorganisation, den Zeitplan, jegliche Planungsunterlagen und die Arbeitsjournale.

Projektaufbauorganisation

TABELLE 1: PROJEKTAUFBAUORGANISATION

Rolle	Abkürzung	Name	E-Mail	Beschreibung
Chefexperte	CEX	Christian Steiner	christian.steiner@kantisargans.ch	Aufsicht über die IPA
Hauptexperte	HEX	Christian Steiner	christian.steiner@kantisargans.ch	Hauptansprechpartner für KAND und VF. Begleitet VF und KAND durch die IPA.
Nebenexperte	NEX	Christian Steiner	christian.steiner@kantisargans.ch	Stellvertretender Ansprechpartner für KAND und VF während der IPA. Begleitet VF und KAND durch die IPA.
Berufsbildner	BB	Christian Steiner	christian.steiner@kantisargans.ch	Verantwortlich für die betriebliche Ausbildung des KAND.
Vorgesetzte Fachkraft	VF	Christian Steiner	christian.steiner@kantisargans.ch	Person, welche dem KAND während der IPA als Ansprechperson zur Verfügung steht. Ansprechperson für die Experten für die jeweilige IPA.
Kandidat	KAND	Paul Pietzko	paul.pietzko@stud.kantisargans.ch	Der Lernende, welcher die IPA durchführt.

Zeitplan

Nr.	Aufgabe	Soll (h)	Ist (h)	20.02.2024	27.02.2024	05.03.2024	12.03.2024	14.03.2024	19.03.2024	02.04.2024	09.04.2024	16.04.2024	23.04.2024	30.04.2024	07.05.2024	14.05.2024
Informationsbeschaffung																
1.0	Dokumente anlegen, strukturieren und sichern	0.2	0.2													
1.1	Theorieeinlage	1	1													
1.2	Aufgabenstellung analysieren	0.3	0.3													
Planung																
2.0	Zeitplan erstellen	1	1													
2.1	Testfälle schreiben	0.5	0.5													
2.2	UML Diagramm designen	0.5	0.5													
2.3	Mockup erstellen	2	2													
Entscheidung																
3.0	Entscheidung fällen	0.2	0.2													
Realisierung																
4.0	Projekt Setup - GitHub, MongoDB	1	1													
4.1	Angular 17, NodeJS	1	1													
4.2	Home Screen	2	2													
4.3	Login Screen + System	4	4													
4.4	Blackjack game loop programmieren	5	7													
4.5	Slots Machines programmieren	4	3													
4.6	Styling	2	2													
Kontrolle																
5.0	Refactoring	2														
5.1	Testing nach Kriterien	2														
Auswertung																
6.0	Auswerten	1														
6.1	Schlusswort	1														
6.2	Kurzfassung	1														
6.3	Dokumentation	4	4													
Sonstiges																
7.0	Arbeitsjournal	2														
7.1	Expertenbesuch	1														
7.2	Ungedachte Aufgaben	1														
7.3	Abgabe	1														
Total		40.7	29.7													
Differenz																

ABBILDUNG 1: ZEITPLAN

Arbeitsjournal

Nachfolgend sind alle Werktage schriftlich in diesem Journal festgehalten. Die im Stundenzettel angegebene Nummer referenziert auf den Zeitplan

Tag 1 – 20.02.2024

Stundenzettel

TABELLE 2: ARBEITSJOURNAL TAG 1 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	14:00	25'	1.0	Dokumente anlegen, strukturieren und sichern
14:00	15:10	70'	1.1	Theorieeinlage
Pause				
15:25	15:55	30'	1.2	Aufgabenstellung analysieren
15:55	16:50	55'	2.0	Zeitplan erstellen
16:50	17:10	20'	2.1	Testfälle schreiben
Pause				
17:20	17:30	20'	7.0	Arbeitsjournal

Saldo

TABELLE 3: ABREITSJOURNAL TAG 1 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8h	3.8h	0	0

Hilfen

TABELLE 4: ABREITSJOURNAL TAG 1 HILFEN

Beschreibung	Quelle(n)
Informationen zu den Anforderungen und Kriterien.	Herr Steiner (EXP)
Layout, Inhalt und Formatierung für Arbeitsjournal, Zeitplan und Testkonzept.	Herr Steiner (EXP)
Informationen zu den einzelnen Testmethoden.	chatGPT

Fragen

TABELLE 5: ABREITSJOURNAL TAG 1 FRAGEN

Thema	Beschreibung	Person
Testen	Ich fragte ob wir testen müssen, wenn wir nicht explizit als Kriterium angegeben haben. Die Antwort war «Ja».	Herr Steiner (EXP)

Reflexion

Mit meiner heutigen Leistung bin ich sehr zufrieden. Nach der heutigen Theorie von Herr Steiner habe ich ein besseres Verständnis, was alles auf uns zukommen wird. Heute habe ich alle

Zukunftsrelevanten Dokumente angelegt. Dies ist eine grosse Hilfe, wenn ich die Dokumente für die richtige IPA in einem Jahr verwenden kann. Denn das Erstellen dieser Dateien hat viel Zeit in Anspruch genommen.

Ausblick

Nächste Woche Dienstag werde ich das UML erstellen und mit der Realisierung der Applikation beginnen. Wobei ich mich zuerst auf die Backendfunktionalität fokussieren werde.

Tag 2 – 27.02.2024

Stundenzettel

TABELLE 6: ARBEITSJOURNAL TAG 2 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	14:05	30'	1.1	Theorieeinlage
14:05	15:05	60'	2.1	UML Diagramm designen
Pause				
15:20	16:30	70'	2.2	Mockup erstellen
16:30	16:50	20'	4.0	Projekt Setup - GitHub, MongoDB
Pause				
17:00	17:40	40'	4.1	Angular 17, NodeJS
17:40	17:50	10'	7.0	Arbeitsjournal

Saldo

TABELLE 7: ABREITSJOURNAL TAG 2 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8h	0	0

Hilfen

TABELLE 8: ABREITSJOURNAL TAG 2 HILFEN

Beschreibung	Quelle(n)
UML Diagramm realisieren	Herr Steiner (EXP)

Fragen

TABELLE 9: ABREITSJOURNAL TAG 2 FRAGEN

Thema	Beschreibung	Person
UML – Planung	Die Frage handelte davon, was alles in den User Stories bzw. den Features abgebildet sein muss.	Herr Steiner (EXP)

Reflexion

Heute war ebenfalls ein erfolgreicher Tag an dem ich viel erledigen konnte. Meine Zeitplanung und meine ursprüngliche Leitfrage 235 (Entwurf mit UWML) musste ich heute etwas umstrukturieren. Anstelle eines UML Diagrammes habe ich ein Floatchart erstellt und ein Mockup 161 (Entwurf, Design) erstellt. Das hat mich heute mehr Zeit gekostet, als eigentlich vorgesehen. Trotzdem konnte ich bereits mit der Realisation der Applikation starten.

Ausblick

Für die nächste Woche ist geplant den Home Screen mit allen dazugehörigen Komponenten (wie bsp. der Nav) und das Login/ Signup System anzufangen und fertig zu stellen.

Tag 3 – 05.03.2024

Stundenzettel

TABELLE 10: ARBEITSJOURNAL TAG 3 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	13:55	20'	1.1	Theorieeinlage
13:55	14:55	120'	4.1	Backend
13:55	15:10	15'	7.1	Expertenbesuch
Pause				
15:25	16:30	65'	6.3	Dokumentation
16:30	16:50	20	4.1	Backend
Pause				
17:00	17:30	30	4.1	Backend

Saldo

TABELLE 11: ARBEITSJOURNAL TAG 3 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	2	1.8	1.8

Hilfen

TABELLE 12: ARBEITSJOURNAL TAG 3 HILFEN

Beschreibung	Quelle(n)
Wichtigkeit der Dokumentation	Herr Steiner (EXP)
Bessere Strukturierung der Arbeit im Ganzen	Herr Steiner (EXP)

Fragen

TABELLE 13: ARBEITSJOURNAL TAG 3 FRAGEN

Thema	Beschreibung	Person
Programmierung	Backendstruktur nach MVC aus vergangenen Projekten	Jonas Schmedtmann

Reflexion

Bei meinem heutigen Expertenbesuch merkte Herr Steiner an, dass ich mit meiner Planung hinterherhinke und mein IPA Bericht oberste Priorität hat, welche ich bis dato noch nicht erstellt hatte. Darum musste ich den Zeitplan auf die neuen Aktivitäten anpassen.

Ausblick

In einer Woche werde ich mich wieder vertieft um die Umsetzung und das Programmieren des eigentlichen Projektes kümmern. Zudem werde ich weiter an meinen Unterlagen feilen.

Tag 4 – 12.03.2024

Stundenzettel

TABELLE 14: ARBEITSJOURNAL TAG 4 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
12:50	13:05	15'	1.1	Theorieeinlage
13:05	15:00	115'	4.1	Backend
Pause				
15:05	15:10	5'	7.0	Arbeitsjournal
Pause				
15:25	14:00	40'	4.1	Backend

Saldo

TABELLE 15: ABREITSJOURNAL TAG 4 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3	0.8	2.6

Hilfen

TABELLE 16: ABREITSJOURNAL TAG 4 HILFEN

Beschreibung	Quelle(n)

Fragen

TABELLE 17: ABREITSJOURNAL TAG 4 FRAGEN

Thema	Beschreibung	Person
Programmierung	Backendstruktur nach MVC aus vergangen Projekten	Jonas Schmedtmann

Reflexion

Mit den heutigen drei, vier Stunden habe ich die Backendlogik verbessert. Auch heute kam ich im Zeitplan nicht so weit voran wie geplant, jedoch ist mir nun klarer, wie und was ich die einzelnen Module aus M294 implementieren muss.

Ausblick

Für die nächsten Lektionen hoffe ich mit dem Backend fertig zu werden. Dies beinhaltet das Login System und das ändern des Kontostandes der User.

Tag 5 – 14.03.2024

Stundenzettel

TABELLE 18: ARBEITSJOURNAL TAG 5 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
7:45	08:05	20'	1.1	Theorieeinlage
08:05	08:35	30'	4.1	Backend
Pause				
08:40	09:20	40'	4.1.0	Backend
Pause				
09:40	12:50	190'	4.1	Backend
Pause				
13:35	15:50	115'	4.4	Blackjack game loop programmieren

Saldo

TABELLE 19: ABREITSJOURNAL TAG 5 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3	0.8	2.6

Hilfen

TABELLE 20: ABREITSJOURNAL TAG 5 HILFEN

Beschreibung	Quelle(n)

Fragen

TABELLE 21: ABREITSJOURNAL TAG 5 FRAGEN

Thema	Beschreibung	Person
Programmierung	Backendstruktur nach MVC aus vergangen Projekten	Jonas Schmedtmann

Reflexion

Mit den heutigen acht Stunden habe ich einiges erreicht. Die Datenbankverbindung habe ich heute endlich erfolgreich fertig gestellt. Und auch bereits an einigen weiteren Features konnte ich bereits weiter- bzw. vorarbeiten. Jedoch ging auch viel Zeit für unnötiges debuggen und stumpfes «Copy & Paste» von ChatGPT drauf. So habe ich den Code unnötig verkompliziert und musste mich erstmal wieder zurecht finden. Und den «Spaghetti code» entwirren.

Ausblick

Für das nächste Mal möchte ich das Login System benutzerfreundlicher machen und mit dem Gamloop des Blackjack Spiels anzufangen. Ausserdem möchte ich die Funktionalität des Abziehens und Hinzufügen der Verluste bzw. Gewinne fertig implementieren.

Tag 6 – 19.03.2024

Stundenzettel

TABELLE 22: ARBEITSJOURNAL TAG 6 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	13:40	5'	1.1	Theorieeinlage
13:40	15:10	90'	4.5	Slots Machines programmieren
Pause				
15:25	17:50	115'	4.40	Blackjack game loop programmieren

Saldo

TABELLE 23: ABREITSJOURNAL TAG 6 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	-2.6	0

Hilfen

TABELLE 24: ABREITSJOURNAL TAG 6 HILFEN

Beschreibung	Quelle(n)

Fragen

TABELLE 25: ABREITSJOURNAL TAG 6 FRAGEN

Thema	Beschreibung	Person
Programmierung	Lösungsansätze zur Programmierung der Funktionalitäten.	OpenAI, Chat GPT
Programmierung	Angular Konventionen	OpenAI, Chat GPT

Reflexion

Diesen Nachmittag konnte ich den Funktionalität des Praktischen Teil dieser Arbeit beinahe fertigstellen. Nach Donnerstag letzter Woche, konnte ich all die angefangen Funktionen miteinander verbinden und zum laufen bringen. Diese Woche bin ich auch viel ruhiger an die Sache herangegen. Was mir dabei half mich mehr in den Programmablauf hineinzusetzen und somit besser zu debuggen.

Ausblick

Für den nächsten Arbeitstag nehme ich mir vor zum einen Die Dokumentation weiterzuschreiben und den geschriebenen Code zu finalisieren. Anschliessend werde ich den code «Refraktoren» und überarbeiten.

Tag 7 – 26.03.2024

Stundenzettel

TABELLE 26: ABREITSJOURNAL TAG 1 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	15:10	95'	6.3	Dokumentation
Pause				
15:25	17:50	115'	6.3	Dokumentation

Saldo

TABELLE 27: ABREITSJOURNAL TAG 7 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	0	0

Hilfen

TABELLE 28: ABREITSJOURNAL TAG 7 HILFEN

Beschreibung	Quelle(n)
Wichtigkeit der Dokumentation	Herr Steiner (EXP)
Allgemeine Verbesserungen in Dokumentationsprozess	Herr Steiner (EXP)

Fragen

TABELLE 29: ABREITSJOURNAL TAG 7 FRAGEN

Thema	Beschreibung	Person

Reflexion

Den Zeitplan habe ich heute nicht eingehalten, da ich den Fokus der heutigen Lektionen mehr auf die Dokumentation legen wollte. Die letzten Nachmittage habe ich alle für das Programmieren meiner Arbeit genutzt. Die nun alle Grundfunktionen korrekt beinhaltet. Für heute wäre eigentlich refactoring angesagt gewesen. Jedoch war ich mit meiner Dokumentation vor heute fast noch nirgends. Zudem hat das zweite Expertengespräch nochmals deutlich gemacht, wie wichtig die Dokumentation zum jetzigen Standpunkt bereits ist. Und darum gingen alle fünf Lektionen dafür drauf. Ich habe fast alles aus der Doku gelöscht und nochmals erfolgreich frisch angefangen.

Ausblick

Beim nächsten Mal werde ich mich wieder an die Dokumentation setzten und versuchen sie weiter zu vervollständigen. Und nach dem ich das Grundgerüst der Dokumentation habe, werde ich mich an das Verfeinern des Codes wagen.

Tag 8 – 02.04.2024

Stundenzettel

TABELLE 30: ABREITSJOURNAL TAG 8 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	15:10	95'	6.3	Dokumentation
Pause				
15:25	16:00	35'	6.3	Dokumentation

Saldo

TABELLE 31: ABREITSJOURNAL TAG 8 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	0	0

Hilfen

TABELLE 32: ABREITSJOURNAL TAG 8 HILFEN

Beschreibung	Quelle(n)

Fragen

TABELLE 33: ABREITSJOURNAL TAG 8 FRAGEN

Thema	Beschreibung	Person

Reflexion

Heute habe ich, wie letzte Woche angekündigt, an der Dokumentation weitergearbeitet. Jedoch ging es mir nach dem Mittag nicht gut, weshalb ich schon um 16:00 Uhr anstatt 17:50 Uhr nach Hause gegangen bin. Diese 110' Minuten werde ich darum zuhause nachholen müssen. Dies ist jedoch nicht allzu schlimm, da für heute nur die Dokumentation geplant war.

Ausblick

Beim nächsten Mal werde ich das Schreiben der Dokumentation pausieren und mich wieder an das Programmieren wagen. Denn die Applikation muss noch nach dem Testprotokoll getestet werden.

Tag 9 – 23.04.2024

Stundenzettel

TABELLE 34: ABREITSJOURNAL TAG 9 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	15:10	95'	6.3	Dokumentation
Pause				
15:25	17:50	145'	6.3	Dokumentation

Saldo

TABELLE 35: ABREITSJOURNAL TAG 9 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	0	0

Hilfen

TABELLE 36: ABREITSJOURNAL TAG 9 HILFEN

Beschreibung	Quelle(n)
Fachwörter lernen	Herr Steiner (EXP)

Fragen

TABELLE 37: ABREITSJOURNAL TAG 9 FRAGEN

Thema	Beschreibung	Person

Reflexion

Die fünf Lektionen habe ich heute wieder komplett in die Erstellung und Nachtragung der Dokumentation gesteckt. Während des heutigen Expertengespräch wurde mir nochmals gesagt, wie wichtig es ist die Dokumentation zeitgleich der Programmierung zu erstellen. Dies habe ich nicht gemacht, da mir die Dokumentation und die Anforderungen an sie zu dem damaligen Zeitpunkt noch zu viel neues auf einmal waren.

Ausblick

Es sind schon 80% der Arbeit vorbei. In der Schule werde ich den Fokus nur noch auf die Dokumentation legen und in Eigenarbeit zuhause die fehlenden Testfälle schreiben und den Code finalisieren. Wobei das nicht im Vordergrund steht.

Tag 10 – 30.04.2024

Stundenzettel

TABELLE 38: ABREITSJOURNAL TAG 10 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	13:55	20'	1.1	Theorieeinlage
13:35	15:10	75'	6.3	Dokumentation
Pause				
15:25	17:50	145'	6.3	Dokumentation

Saldo

TABELLE 39: ABREITSJOURNAL TAG 10 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	0	0

Hilfen

TABELLE 40: ABREITSJOURNAL TAG 10 HILFEN TABELLE 41: ABREITSJOURNAL TAG 11 HILFEN

Beschreibung	Quelle(n)
Fachwörter lernen	Herr Steiner (EXP)

Fragen

TABELLE 42: ABREITSJOURNAL TAG 01 FRAGEN

Thema	Beschreibung	Person

Reflexion

Wie letzte Woche beschlossen, habe ich mich auch heute wieder alleine der Doku gewidmet. Diese habe ich sehr weit vorantreiben können. So bin ich bis zur Realisierung vorangeschritten. Ich habe es sogar geschafft die gesamte Backendlogik zu Papier zu bringen und simpel zu erklären.

Ausblick

Da nächste Woche bereits die Abgabe stattfinden wird, bin ich gezwungen am Wochenende weiter zuarbeiten. Ich denke nicht dass ich es alleine in den der Schule schaffen würde die Doku fertig zu stellen und die Software wie geplant zu testen.

Tag 11 – 23.04.2024

Stundenzettel

TABELLE 43: ABREITSJOURNAL TAG 11 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	15:10	95'	6.3	Dokumentation
Pause				
15:25	17:50	145'	6.3	Dokumentation

Saldo

TABELLE 44: ABREITSJOURNAL TAG 11 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	0	0

Hilfen

TABELLE 45: ABREITSJOURNAL TAG 11 HILFEN

Beschreibung	Quelle(n)
Fachwörter lernen	Herr Steiner (EXP)

Fragen

TABELLE 46: ABREITSJOURNAL TAG 11 FRAGEN

Thema	Beschreibung	Person

Reflexion

Die fünf Lektionen habe ich heute wieder komplett in die Erstellung und Nachtragung der Dokumentation gesteckt. Während des heutigen Expertengespräch wurde mir nochmals gesagt, wie wichtig es ist die Dokumentation zeitgleich der Programmierung zu erstellen. Dies habe ich nicht gemacht, da mir die Dokumentation und die Anforderungen an sie zu dem damaligen Zeitpunkt noch zu viel neues auf einmal waren.

Ausblick

Es sind schon 80% der Arbeit vorbei. In der Schule werde ich den Fokus nur noch auf die Dokumentation legen und in Eigenarbeit zuhause die fehlenden Testfälle schreiben und den Code finalisieren. Wobei das nicht im Vordergrund steht.

Tag 12 – 05.05.2024

Stundenzettel

TABELLE 47: ABREITSJOURNAL TAG 12 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
10:00	12:30	159'	6.3	Dokumentation
Pause				
14:00	16:00	120'	6.3	Dokumentation

Saldo

TABELLE 48: ABREITSJOURNAL TAG 12 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	0	0

Hilfen

TABELLE 49: ABREITSJOURNAL TAG 12 HILFEN

Beschreibung	Quelle(n)
Fachwörter lernen	Herr Steiner (EXP)

Fragen

TABELLE 50: ABREITSJOURNAL TAG 12 FRAGEN

Thema	Beschreibung	Person

Reflexion

Wie letzte Woche beschlossen, habe ich mich auch heute wieder alleine der Doku gewidmet. Diese habe ich sehr weit vorantreiben können. So bin ich bis zur Realisierung vorangeschritten. Ich habe es sogar geschafft die gesamte Backendlogik zu Papier zu bringen und simpel zu erklären.

Ausblick

Da nächste Woche bereits die Abgabe stattfinden wird, bin ich gezwungen am Wochenende weiter zuarbeiten. Ich denke nicht dass ich es alleine in den der Schule schaffen würde die Doku fertig zu stellen und die Software wie geplant zu testen.

Tag 13 – 07.05.2024

Stundenzettel

TABELLE 51: ABREITSJOURNAL TAG 13 STUNDENZETTEL

Start-zeit	End-zeit	Zeit	Nr.	Aufgabe
13:35	13:55	20'	1.1	Theorieeinlage
13:35	15:10	75'	6.3	Dokumentation
Pause				

Saldo

TABELLE 52: ABREITSJOURNAL TAG 13 SALDO

Soll	Ist	Tagessaldo	Saldo total
3.8	3.8	0	0

Hilfen

TABELLE 53: ABREITSJOURNAL TAG 13 HILFEN

Beschreibung	Quelle(n)

Fragen

TABELLE 54: ABREITSJOURNAL TAG 13 FRAGEN

Thema	Beschreibung	Person

Reflexion

Ausblick

Teil 2

In diesem zweiten Teil der Dokumentation wird das Projekt von der Aufgabenstellung bis zur Auswertung beschrieben.

Projektmanagementmethode

Eine passende Projektmanagementmethode ist die Voraussetzung einer effizienten Entwicklung. Die gängigsten Varianten und die von mir benutzte werden unten aufgelistet.

Vorgehensmodelle können in iterative und sequenzielle Arten unterteilt werden. Iterative Modelle, wie beispielsweise «Scrum», werden in der Softwareentwicklung meist bevorzugt. Sie ermöglichen die schrittweise Entwicklung und Auslieferung von kleinen Teilen der Software in kurzen Zyklen. Hat man aber zu Beginn bereits ein vollständiges Ziel, arbeitet allein oder will den Fokus auf die Dokumentation legen, ist dies wohl nicht der ideale Ansatz. In solchen Fällen wählt man besser eines der sequenziellen Vorgehensmodelle, bei welchen Wert auf klare aufeinanderfolgende Phasen gelegt wird. Dabei wird mit der nächsten Tätigkeit erst begonnen, wenn der vorherige Schritt vollständig abgeschlossen ist. Im Fokus steht eine intensive Planungsphase, welche Unklarheiten und Veränderungen während der Umsetzung minimieren soll. Auch eine strenge Kontrolle und hochwertige Dokumentation wird bei den meisten dieser Modelle ermöglicht.

Die Entscheidung fiel auf das sequenzielle IPERKA Modell. Wenn sauber nach IPERKA gearbeitet wird, können alle Punkte aus der Vorgabe abgedeckt werden.

Bei diesem Modell wird ein Projekt in sechs Phasen unterteilt:

- **Informieren:** In einem ersten Schritt analysiert man den Auftrag, sodass man diesen genauestens versteht.
- **Planen:** Daraufhin beginnt man sich Gedanken über mögliche Lösungsvarianten zu machen.
- **Entscheiden:** Danach fällt man Entscheidungen zu den einzelnen Varianten.
- **Realisieren:** Im vierten Schritt beginnt man mit der Umsetzung des Projektes.
- **Kontrollieren:** Später wird überprüft, ob der Auftrag korrekt umgesetzt wurde.
- **Auswerten:** Als Letztes wird reflektiert, wie die Arbeit verlief.

Ein Nachteil hat IPERKA. Fehler werden erst nach der kompletten Umsetzung erkannt. So ist es schwierig, auf diese zu reagieren. Deshalb ist es ratsam schon während der Realisierung die einzelnen Komponenten zu testen, bevor sie als erledigt angesehen werden. In der Kontrollphase wird dann die Applikation im Komplettpaket auf alle Kriterien überprüft.

Neben IPERKA gibt es auch andere sequenzielle Projektmanagementmethoden. Das Wasserfall- oder V-Modell wären zwei grosse Vertreter solcher Managementmethoden. Da jedoch IPERKA häufiger in schulischen Zwecken verwendet wird. Findet sie nun auch hier besseren Anklang.

1 Informationsbeschaffung

Als Erstes musste verstanden werden, warum es nötig ist den Auftrag zu erfüllen. Dies ermöglicht es, die Sicht des Auftraggebers besser nachzuvollziehen.

Online Casinos sehen von Aussen betrachtet sehr komplex und eindrucksvoll aus, sind jedoch in ihrem Kern einfach umzusetzen. Zudem ist es ein perfektes Projekt für eine Multiuserapplikation mit festgelegten Zielen und nicht Zielen. Die Applikation wurde mit dem JavaScript Framework Angular umgesetzt, da dies ein sehr robustes Framework ist. Die App beinhaltet keine komplexe Animationen, weshalb sie gut mit Angular umzusetzen ist.

1.1 Erwartbarkeit

Die Arbeit soll starken Wert auf die Umsetzung mit Clean Code, Kiss und weiteren coding Konventionen wie MVP setzen. Codedateien sollen in eine verständlich gute Struktur unterteilt werden.

1.2 Ziel

All die gelernten coding Konventionen aus den drei Jahren Informatik Unterricht an der Kantonschule Sargans sollen sauber umgesetzt werden, so dass der Code auch in Zukunft gut wartbar und erweiterbar ist.

1.3 Unklarheiten

Wie soll die Backendstruktur aussehen. Zuerst war eine verknüpfte Backend mit Frontend Variante angedacht. Wobei Die SSR Funktionalität von Angular so genutzt wird, dass alles auf einem Server läuft.

Jedoch wurde durch die Leitfrage 249 – MVC (Programmierung) diese Überlegung über den Haufen geworden.

1.4 Projektumfeld

Das Projektumfeld beinhaltet die Gesamtheit aller internen und externen Faktoren, die das Projekt beeinflussen können. Es umfasst die organisatorischen Rahmenbedingungen, die technische Infrastruktur, die beteiligten Personen und Gruppen, deren Interessen und Einflüsse, sowie die relevanten Markt- und Umweltbedingungen.

1.4.1 Abgrenzung

Die folgend aufgelisteten Themen gehören nicht zum Umfeld dieser IPA Arbeit.

Design

Das Design wurde nicht von Grund auf neu erfunden, sondern wurde von bekannten Casino Seiten wie «Jackpots.ch» abgeleitet.

Assets

Einige Illustrationen wie beispielsweise die Spielkarten SVGs wurden von GitHub repositories entnommen, da das erstellen solcher Dateien unnötig Zeit in Anspruch nehmen würde.

Testing Systeme

Die eingesetzten Test-Frameworks Jasmine und Karma wurden eingesetzt schneller und qualitativ besser testen zu können.

1.4.2 Umsysteme

Einige Funktionen wurden als library als npm install in das Projekt heruntergeladen. Bcrypt und JWT sind Beispiele, die in Eigenentwicklung ebenfalls zu viel Zeit in Anspruch genommen hätten den Rahmen dieser Arbeit masslos gesprengt hätten.

1.4.3 Assets

Für dieses Projekt wurde eine SVG Sammlung von Pokerkarten importiert. Zudem wurde in Eigen Kreation eine Sammlung von Jeton

1.4.4 Visual Studio Code

Visual Studio Code wurde als IDE verwendet, da sie Industrie Standard ist.

Dazu wurden einige Add-Ons in VS Code installiert um den Entwicklungsprozess zu vereinfachen. Dazu gehören grössten Teils Add-Ons von den Entwicklern der Technologien, wie MongoDB, Angular, NodeJS usw.

1.4.5 GitHub

Zur Versionierung, Synchronisierung und Aufbewahrung des Codes dient GitHub für dieses Projekt. GitHub wird ebenfalls an der KSS als Versionsverwaltungstool standardmässig eingesetzt.

Während der ganzen Entwicklung wird nur auf dem Main Branch gearbeitet, da das Projekt von nur einer Person entwickelt wird und nicht veröffentlicht wird.

1.4.5 MongoDB

Für das Speichern von Nutzerdaten dient MongoDB; eine weit verbreitete Datenbanklösung welche das Speichern in JSON Format ermöglicht, was wiederum mit der Programmierung von JavaScript harmoniert.

1.4.6 Postman

Als Testmittel der zu erstellenden REST-API wird Postman eingesetzt. Dies ermöglicht es per JSON Anfragen an einen definierten API-End Point Anfragen zu senden und Responses auf dessen Korrektheit zu prüfen.

1.4.7 Konventionen

Im Auftrag wurde beschrieben, dass die allgemeinen wie auch zusätzliche Leitfragen Konventionen eingehalten werden sollen, um den im Projektauftrag beschrieben Clean Code Standard zu halten.

Struktur

```
backend
|- controllers
|  |- authController.js
|  |- errorController.js
|  |- userController.js
|
|- models
|  |- userModel.js
|
|- routes
|  |- userRoutes.js
|
|- utils
|  |- appError.js
|  |- app.js
|  |- server.js
```

ABBILDUNG 2: BACKEND
STRUKTUR

```
app
|- components
|  |- home
|  |- login
|  |- navbar
|  |- signup
|  |- welcome
|
|- games
|  |- blackjack
|  |- slots
|
|- models
|
|- services
|  |- 3Dcard.service.ts
|  |- auth.service.ts
|  |- blackjack.service.ts
|  |- jeton.service.ts
|  |- slots.service.ts
|
|- app.component.html
|- app.component.scss
|- app.component.spec.ts
|- app.component.ts
|- app.config.server.ts
|- app.config.ts
|- app.routes.ts
```

ABBILDUNG 2: BACKEND
STRUKTUR

Commits

Commits werden immer nach dem Datum des gemachten Tages benannt. Dabei wird immer am Ende des Arbeitstages committed, es sei denn es gibt viel Code der auf einmal geändert wird. Dann werden Mehrere Commits an einem Tag gemacht, welche mit v +Nummer (v1, v2..) benannt werden.

Beispiel:

26.03.2024 v1

Branches

Der ganze Entwicklungsprozess findet alleine auf dem Default Main Branch statt.

1.5 Besprechung

Um sicherzustellen, dass alle Anforderungen an das Projekt korrekt formuliert waren, wurde das Projekt abwechselnd mit dem HEX besprochen und angepasst.

2 Planung

2.1 Zeitplan

Nach der Informationsbeschaffung und einer genauen Analyse konnte der Zeitplan erstellt werden. Das Gantt Diagramm ist Teil 1 unter «Zeitplan» ersichtlich.

2.1.1 Rundung

Vor der Erstellung wurde definiert, wie detailliert die Aufgaben geplant werden sollen. Die Aufteilung besteht jeweils aus Paketen, die jeweils nicht länger als vier Stunden sein dürfen. Da es aber auch viele kleine Aufgaben gibt, wurden diese manche mit anderen Aufgaben zusammengefügt und dessen Zeitaufwand nach bestem Wissen geschätzt.

2.1.2 Interpretation

Für die Aufgaben während der Realisierung wurde Zeit für die Dokumentation inkludiert. Wenn also für eine Funktion 3 Stunden geplant wurden, darf ein Teil dieser Zeit auch zur Dokumentierung des Vorgehens genutzt werden.

2.1.3 Blocker

Für die Informationsbeschaffung und Erstellung des Zeitplans wurde keine genaue Zeit geschätzt. Dies liegt an IPERKA. Schliesslich konnte erst bei der Planungsphase geplant werden. Zu Beginn wurden aber extra Zeit blockiert.

2.1.4 Ist-Zeit

An jedem Arbeitstag werde ich am Schluss die Zeiten eintragen und einen Soll-Ist-Vergleich vornehmen. Dieser wird dann im Arbeitsjournal festgehalten. Die im Arbeitsjournal eingetragenen Zeiten werden immer auf fünf Minuten gerundet und in Minuten angegeben.

2.1.5 Meilensteine

Diese Meilensteine wurden zu Beginn der Arbeit definiert. Um den Soll-Ist-Vergleich besser abgleichen zu können.

TABELLE 55: MEILENSTEINE

Meilenstein	Tag der geplanten Erreichung	Tag der Erreichung
Angular Anwendung Mit allen Screens	27.02	27.02
Rest API mit allen Endpoints	05.03	14.03
Blackjack Game Loop	12.03	14.03
Slots Maschinen Game Loop	14.03	26.03
Refactoring	26.03	16.04
Dokumentation	16.04	07.05

2.2 Testkonzept

Getestet wird, um sicherzustellen, dass eine Software-Anwendung korrekt funktioniert, Benutzeranforderungen erfüllt und frei von Fehlern ist, die die Nutzung beeinträchtigen könnten, sowie um das Risiko von Problemen nach der Veröffentlichung zu minimieren und dadurch die Qualität und Zuverlässigkeit des Produktes zu gewährleisten.

2.2.1 Testmethoden

- **Blackbox-Test:** Hierbei kennt der Tester den internen Aufbau oder den Code der Anwendung nicht und testet nur die Benutzeroberfläche und die Funktionalitäten, um sicherzustellen, dass die Anforderungen erfüllt sind.
- **Whitebox-Test:** Im Gegensatz zum Blackbox-Test hat der Tester Zugang zum internen Code und kann so die internen Strukturen testen, um die Abdeckung des Codes sicherzustellen, beispielsweise durch Pfad- oder Zweigtests.
- **Unit-Test:** Dabei werden einzelne Komponenten oder Funktionen der Software isoliert getestet, um ihre korrekte Funktionsweise zu verifizieren.
- **Integrationstest:** Hierbei werden mehrere Komponenten oder Systeme kombiniert und als Gruppe getestet, um zu überprüfen, ob sie korrekt zusammenarbeiten.
- **Systemtest:** Bei dieser Art des Tests wird das gesamte, vollständig integrierte System auf Übereinstimmung mit den spezifizierten Anforderungen geprüft.
- **Akzeptanztest:** Dieser Test wird durchgeführt, um festzustellen, ob das System die Geschäftsanforderungen erfüllt und für den Einsatz beim Endbenutzer bereit ist.

Jasmine

Mit Jasmine können Behavior-Driven Development (BDD) Tests für JavaScript-Anwendungen erstellt werden. Es legt Wert darauf, dass das Verhalten einer Funktion genau wie spezifiziert überprüft wird. Einmal implementiert, helfen die Tests Entwicklern zu verstehen, ob das Verhalten der Funktion auch nach Änderungen noch dem vorgegebenen Muster entspricht. Weitere Informationen zu Jasmine sind auf der Webseite: <https://jasmine.github.io/> verfügbar.

Karma

Karma ist ein Test-Runner, der für das Ausführen von JavaScript-Tests direkt im Browser entwickelt wurde. Dies ermöglicht es, dass Tests unter realen Bedingungen ausgeführt werden können. Nachdem die Testfälle implementiert sind, wird überprüft, ob die Ergebnisse den Erwartungen entsprechen, was gerade bei Cross-Browser-Testing von Vorteil ist. Mehr zu Karma findet man unter: <https://karma-runner.github.io/>.

Whitebox

Beim White-Box-Test werden interne Strukturen einer Anwendung getestet. Hierbei geht es darum, die internen Wege, die Logik und die Implementierungsdetails des Codes zu überprüfen. Ein Test wird so implementiert, dass er die interne Funktionsweise überprüfen und sicherstellen kann, dass alle Pfade und Zweige des Codes unter verschiedenen Bedingungen funktionieren. Dies erfordert tiefes Verständnis des zu testenden Systems.

Schlussfolgerung

Wie aus den Test-Methoden bzw Test Frameworks ersichtlich wird. Wird das Projekt auf zwei Arten getestet. Zum einen Automatische Test, die direkt im Browser ausgeführt werden, als auch per Hand aus Entwicklersicht.

- **Automatisch**
Manche Test werden vom Programm während der Laufzeit automatisch ausgeführt. Diese werden im Vorfeld definiert und führen sich dann von alleine immer wieder selbst aus.
- **Manuell**
Zudem werde ich einzelnen Funktionen per Hand im ausgeführten Programm destruktiv testen.

Bei jedem Testfall ist in der Spalte «Art» jeweils einer dieser zwei Begriffe angegeben. Dadurch ist nachvollziehbar, in welcher Umgebung die Tests durchgeführt wurden.

Hinweis: Details zum Vorgehen werden nicht getestet. So werden beispielsweise die Einhaltung der Commit Conventions nicht getestet, da diese später unter «Auswertung» überprüft werden.

2.2.2 Testdaten

Es werden die erhobenen Daten der Benutzer ausgewertet. Ansonsten muss nichts beachtet werden.

2.2.3 Testprotokoll

Getestet wird auf Windows 11 mit Opera als Browser. Dabei werden die Testing-Frameworks Jasmine in der Version 3.9.0 und Karma in der Version 6.3.2 Verwendet. Destruktive Whitebox Tests werden sowohl im Frontend als auch im Code durchgeführt. Dabei wird die Datenbank des Projektes MongoDB auf ihre Korrektheit geprüft. Der zu testende Code im Backend ist Node.js in der Version 14.17.0 und im Frontend Angular in der Version 17.0.0. Der Tester wird dabei der Kandidat KAND sein.

2.3 Testfälle

Nachfolgenden sind alle Testfälle definiert. Diese decken alle relevanten Kriterien des Auftrages ab.

Basisarbeiten

TABELLE 56: TESTKONZEPT TESTFÄLLE

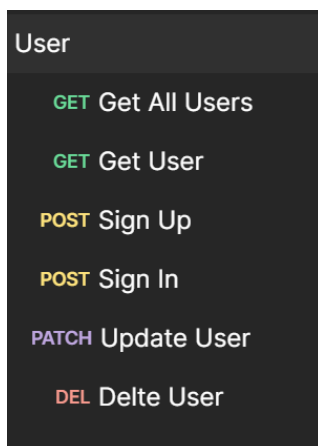
Nr.	Art	Was wird getestet?	Erwartetes Ergebnis
1.0	Whitebox-Tests (Jasmine)	Registrierung eines neuen Benutzers	Eingabe von validen Benutzerdaten
2.0	Whitebox-Tests (Jasmine)	Login-Funktionalität	Eingabe von gültigem Nutzernamen und Passwort
3.0	Whitebox-Tests (Jasmine)	Aktualisierung des Kontostandes	Ergebnis eines Slotmaschinenspiels (Gewinn oder Verlust)
4.0	Unit-Tests (Karma)	Ergebnisse eines Slotmaschinen-Spins	Durchführung eines Spins mit Einsatz
5.0	Unit-Tests (Karma)	Spiellogik von Blackjack	Durchführung einer Hand Blackjack
6.0	Unit-Tests (Karma)	Validierung von Benutzerdaten bei Registrierung	Eingabe von fehlerhaften Benutzerdaten

2.5 Datensicherung und Versionierung

Es ist wichtig, dass Daten vor Verlust geschützt sind. Zudem ist auch eine Versionierung von Nöten, um auf älteren Code zugreifen zu können. Der Sourcecode wird dementsprechend auf GitHub abgespeichert und ist so versioniert und protokolliert. Lokale Änderungen werden dazu zu sinnvollen Zeitpunkten gepusht. Die Dokumentation und der Zeitplan werden ebenfalls auf Vorgabe des HEX in das GitHub Repository hochgeladen.

2.6 Rest API

Um die Kommunikation zwischen der Anwendung und der Datenbank zu gewährleisten wurde mithilfe von Node.js eine RestAPI erstellt. Die API beinhaltet die folgenden Endpunkte und Funktionen. Siehe Abbildung 2.



User
GET Get All Users
GET Get User
POST Sign Up
POST Sign In
PATCH Update User
DEL Delete User

ABBILDUNG 3: API ENDPOINTS

2.6.1 JWT

JSON Web Token ist ein offener Standard, der eine kompakte und selbständige Methode für sicher übermittelte Informationen zwischen Parteien als JSON-Objekt bietet. In dieser Anwendung werden JWTs verwendet, um die Authentifizierung von Benutzern zu verwalten.

2.6.2 Fehlerbehebung

Eine API ist eine weitere Abhängigkeit, die korrekt konfiguriert sein muss, da sonst die Applikation im ganzen nicht funktioniert. Die Applikation würde unter Umständen bei Ausfällen der API nicht funktionieren. Dies ist unter folgenden Umständen der Fall:

- Bei einem Ausfall des hostenden Servers
- Falls die Internetverbindung des Rechners ausfällt

In solchen Fällen müssen den Benutzern entsprechende Fehlermeldungen angezeigt werden.

2.6.3 Mock Daten

Um während der Entwicklung den geschriebenen Code testen zu können wurden einige fiktive Benutzer angelegt. Diesen wurde über die Zeit immer wieder Geld übertragen, um die Gewinne und Verluste überprüfen zu können. Mockdaten sind vorgefertigte Werte, welche genutzt werden können, um ein Verhalten zu simulieren.

2.7 Daten

Da die Applikation Nutzerdaten sammelt um funktionieren zu können, stellt sich die Frage, wo diese gelagert werden und wie man ihre Sicherheit garantieren kann.

2.7.1 Public Ordner

In Angular gibt es den «public» Ordner. Dieser ist angedacht um leifht pflegbare Daten, wie Illustrationen und andere Frontendwichtige Dateien zu lagern. Diese Daten werden dann mit der Applikation als ganzes auf einen fremden Server Deployed, wenn die Absicht besteht, die Applikation zu veröffentlichen.

MongoDB

Die gesamten gesammelten Nutzerdaten werden über das erstellte Backend an das verbundene MongoDB Cluster gesendet. Dort werden sie in der Collection «Users» in der «data» Datenbank gespeichert.

Sicherheit

Die Passwörter werden nur hashed gespeichert. Dies verhindert unerlaubten Zugriff auf Nutzerkonten nach einer Attacke auf die Applikation.

2.8 Architektur

Dieses Projekt hält sich an die MVC (Model, View, Controller) Architektur. Dazu wird die Software in drei Schichten «Model», «View» und «Architektur» unterteilt. Jede Schicht ist für eine Aufgabe zuständig und kommuniziert mit der direkt darüber- oder darunterliegenden Schicht. Dadurch werden die Verantwortlichkeiten klar getrennt, weshalb ganze Schichten auch leicht ausgetauscht werden können.

Das Backend wird dabei in Models und Controllers unterteilt. Jedoch dient das Backend nur als API also als Schnittstelle von Datenbank zum Frontend. So wird im Angular Code nicht nur das Styling sondern auch die Logik der Applikation gehandhabt. In Angular Selber wird dabei auf die Konventionen zurückgegriffen die beim Umgang mit dem Framework üblich sind.

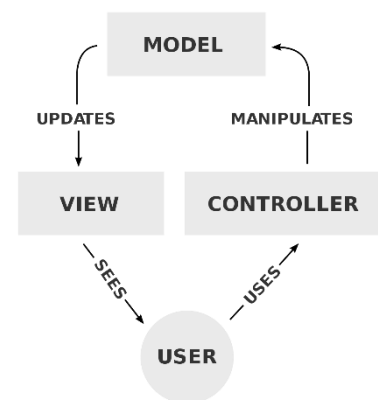


ABBILDUNG 4: MVC ARCHITEKTUR

2.8.1 Frontend

Für die Umsetzung des Frontends wurde folgender Ablauf geplant. In den unten stehenden Abbildungen sind die einzelnen Funktionen und Gameloops definiert bzw. abstrahiert.

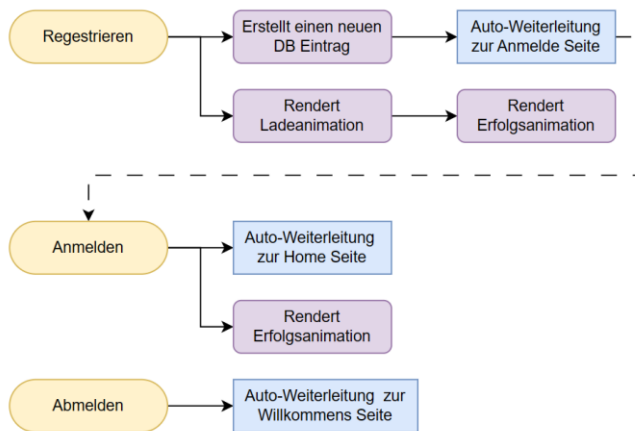


ABBILDUNG 7: REGISTRIERUNG ARCHITEKTUR

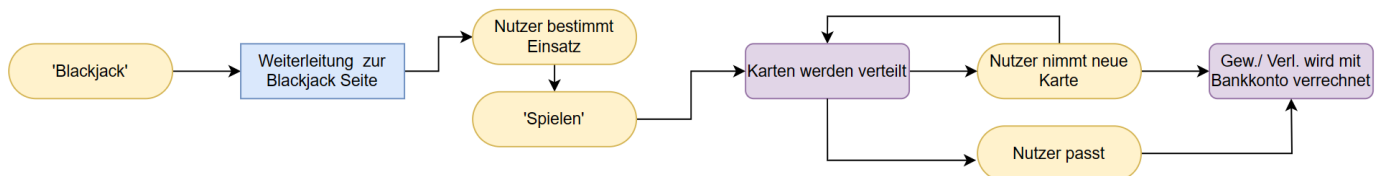


ABBILDUNG 6: BLACKJACK GAME LOOP ARCHITEKTUR

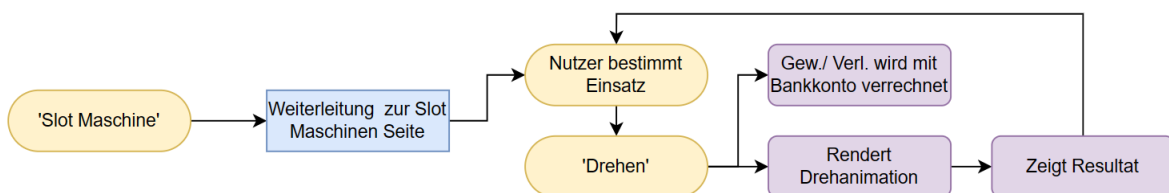


ABBILDUNG 5: SLOT MASCHINE ARCHITEKTUR

2.9 Node Version

Wichtig bei der Installation ist, dass eine kompatible Node-Version verwendet wird. Die Anforderungen in diesem Projekt sind Versionen ab 18.13.0 und neuer unterstützt.

2.10

In diesem Projekt werden folgende Schriften verwendet:

- Roberto
- Arial

3 Entscheidung

In der dritten Phase von IPERKA werden Entscheidungen getroffen.

3.1 Angular Setup

Wie ein Angular Projekt generiert werden kann, steht in folgender Anleitung beschrieben:

<https://angular.io/tutorial/tour-of-heroes/toh-pt0>

Bei der Initialisierung dieses Projektes wurden folgende Schritte getätigt:

TABELLE 57: ANGULAR INITIALISIERUNGSFRAGEN

Abfrage	Erklärung	Möglichkeiten	Entscheid	Begründung
Preferred stylesheet format	Verschiedene Arten von CSS stehen zur Auswahl	CSS, SCSS, Sass, Less	SCSS	Leicht verständlich und Schlanke als basis CSS
ServerSide Rendering	Soll die Applikation Server Side gerendert werden	Yes/ No	Yes	ServerSide Rendering hat viele vorteile gegenü. ClientSide Rendering

3.2 Packages

Als Programmierer muss man nicht alles von Grund auf selbst entwickeln. Es gibt Codegerüste, welche ein anderer Programmierer bereits vollends ausprogrammiert hat und anderen in der Branche gratis zur Verfügung stellt.

Dieses Projekt beinhaltet Packages von mehreren externen Technologien. Vermehrt sind Angular spezifische packages installiert, die zum Teil vorausgesetzt sind und zum anderen viele Aufgaben vereinfachen.

Zudem sind Backend spezifische packages wie cors, dotenv, morgen und express wie auch mongoose für die Datenbank vorhanden.

Um den Code besser abstrahieren zu können, wird RxJS verwendet. Diese library ermöglicht es Daten in einer deklarativen Art zu manipulieren.

3.3 Daten

In der Planung wurden Methoden zu Speicherung von Daten aufgeführt. In diesem Abschnitt wird nun der Entscheid für die Implementierung der Route dargelegt. Unter «api/v1/users» werden die Nutzerdaten für das Frontend, mit erfolgreicher Authentifizierung, als JSON zur Verfügung stehen.

Es stellt sich jedoch noch die Frage, ob alle Daten auf einmal gefetchet und in den Komponenten manipuliert oder der Service die Logik bei sich hält und nur bestimmte Daten zurükliefert. Die Lösung sind unterschiedliche Endpoints von der API. Die entsprechenden Komponenten, fetchen also immer nur die Daten, die sie auch wirklich brauchen. So kann ein Overlord von Daten unterbunden und die Leistung gesteigert werden.

3.5 Sonstige Konventionen

Im Projekt werden alle von Angular geforderten Konventionen korrekt umgesetzt. So werden einzelne Komponenten je nach größe weiter unterteilt und einzelne Funktionen in Services outsourcet .

4 Realisierung

Nach dem die Entscheidungen getroffen wurde, konnte die Planung realisiert werden.

4.0.1

Um die Applikation zu starten muss man die Angular CLI lokal installiert haben. Danach führt man im Terminal den Befehl: `ng serve` aus.

Um das Backend zu starten muss Node und NPM ebenfalls lokal installiert sein. Danach navigiert man zum «backend» Folder `cd src/backend`. Um das Backend zu starten führt man dann den Befehl: `node server.js` aus.

4.1 Erstellung Git Repository

Mit der Initialisierung einer Angular Anwendung wird automatisch ein lokales Git Repo erstellt. Dieses kann dann entweder durch GitHub Desktop oder direkt über Visual Studio Code direkt auf GitHub veröffentlicht werden. Die Sichtbarkeit dieses Projektes ist Privat, wobei der HEX als Kollaborator auf das Projekt Einsicht hat. Zudem werden die Dokumente wie der Zeitplan und der IPA-Bericht jede Woche, entsprechend den commit Konventionen, mit dem Code gepusht.

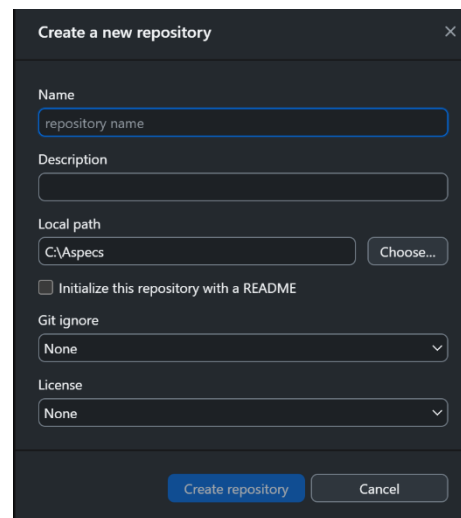


ABBILDUNG 8: ERSTELLUNG REPOSITORY

Nachdem das Projekt erstellt wurde, wurde wie beschrieben Der HEX als Kollaborator dem Projekt hinzugefügt.

Danach konnte das Repository im Terminal mit dem folgenden Befehl geklont werden:

```
git clone https://github.com/paulpietzk/casino-app.git
```

4.2 Angular Setup

Das Setup wurde wie geplant durchgeführt. Dazu waren vier Schritte notwendig.

4.2.1 Node Version auswählen.

Bei der Initialisierung wurde automatisch die neueste (@Latest) kompatible Version von Node ausgewählt.

4.2.2 Projekt Generieren

Das Projekt wurde dann entsprechend dem beschriebenen Setup Prozess (siehe 3.1 Angular Setup) generiert.

4.2.3 Pakete installieren

Wie im Kapitel «Entscheidungen» beschrieben wird, werden noch weitere NPM-Pakete im Projekt benötigt. Dieser Befehle installiert beispielsweise mongoose `npm i mongoose`.

4.2.4 Aufräumen

Zum Schluss musste das Projekt noch etwas aufgeräumt werden. Schliesslich sollen keine irrelevanten Dateien und Konfigurationen das Repository überfüllen.

Um den Projektumfang nicht zu gross zu machen, wurde im Projekt nur unbedingt notwendige libraries und Operationen verwendet bzw. installiert.

4.3 Linting und Testing

Das Ziel ist, einen gleichmässigen Code zu schreiben. Dafür müssen Regeln definiert werden, nach denen sich die Entwickler richten müssen. Dies ist die Voraussetzung für einen sauberen Code. Beim Linting während des Deploy-prozesses überprüft, ob diese Regeln eingehalten wurden.

Jedoch wurde in diesem Projekt kein spezifischer Lintner verwendet, da aus Zeit- und Managementgründen der Frontend Code mit dem Backendcode zusammengelegt wurde. Durch den EXP wurde festgelegt, dass das Projekt nur auf einem Repository angelegt werden soll. Durch diese Struktur liessen sich keine Lintner konfigurieren, da diese entweder für den Backendcode (Node) oder für das Frontend (Angular) konfiguriert werden müssen.

Folgende Lintner und CI/CD Pipeline Tools wären mit einer anderen Projektstruktur kompatibel.

TABELLE 58: CI TOOLS

Tool	Beschreibung
ESLint	Analysiert JavaScript, TypeScript; erzwingt Code-Standards
Staged Lintner	Prüft geänderte Dateien vor Commit; verwendet Husky, lint-staged

4.3.1 Prettier

Prettier ist dazu da, den Code nach Vorgaben zu formatieren. Mit dem Befehl `npm run prettier` wird der Gesamte code geprüft und angepasst. Mehr über das Tool erfährt man auf der offiziellen Website: <https://prettier.io/>. Dieses Tool nimmt direkte Änderungen am Code vor. Dabei wird jede Zeile

nach den Vorgaben formatiert. Beispielsweise wird jeder Zeile ein Semikolon angefügt, wenn man dies vorgibt

```
> casino-app@0.0.0 prettier
> prettier --write .

.vscode/extensions.json 54ms (unchanged)
.vscode/launch.json 7ms (unchanged)
.vscode/tasks.json 10ms (unchanged)
angular.json 17ms
eslint.config.mjs 13ms
package-lock.json 226ms (unchanged)
package.json 2ms (unchanged)
README.md 53ms (unchanged)
server.ts 302ms
src/app/app.component.html 38ms
src/app/app.component.scss 25ms (unchanged)
src/app/app.component.spec.ts 15ms
src/app/app.component.ts 11ms (unchanged)
src/app/app.config.server.ts 6ms
src/app/app.config.ts 6ms (unchanged)
src/app/app.routes.ts 7ms
src/app/components/home/home.component.html 26ms
src/app/components/home/home.component.scss 39ms
src/app/components/home/home.component.spec.ts 10ms
src/app/components/home/home.component.ts 25ms
src/app/components/login/login.component.html 28ms
src/app/components/login/login.component.scss 6ms
src/app/components/login/login.component.spec.ts 11ms
src/app/components/login/login.component.ts 20ms
src/app/components/navbar/navbar.component.html 10ms (unchanged)
```

ABBILDUNG 9: PRETTIER

4.5 CI/CD

Continuous Integration/ Continuous Deployment ist ein Prozess, bei welchem der Code regelmässig und automatisch getestet, gebildet und auf ein Repository oder sogar eine Produktionsumgebung geladen wird. Durch die Automatisierung wird sichergestellt, dass der Code jederzeit fehlerfrei ist.

Aus Zeitgründen wurde auf eine CI Pipeline verzichtet. Und da das Projekt nicht real deployed wird, konnte auch auf eine CD Pipeline verzichtet werden.

4.6 Branches

Da nicht geplant war auf einem anderen Branch als dem Main Branch zu arbeiten, konnte das Git direkt benutzt werden. Dies war möglich, da nur der KAND alleine an dem Projekt gearbeitet hat und keine CD/CD Integration vorhanden war.

4.7 Backend

Während des Informatikunterricht an der IMS wurden alle Web Applikationen mit ein und der gleichen Struktur nach MVC programmiert. Deshalb wurde diese Struktur in diesem Projekt übernommen.

4.7.1 Server.JS

Dabei wird in der server.js eine einfache Verbindung zum MongoDB Cluster erstellt.

4.7.2 App.JS

In der app.js wird die Middleware definiert. Und beispielsweise Cors für alle Endpoints aktiviert.

4.7.3 AppError

Im «utils» folder wird eine benutzerdefinierte Fehlerklasse AppError definiert, die von der eingebauten JavaScript-Klasse Error erbt.

4.7.4 Routes

Unter «routes» werden die Routen für Benutzeroperationen definiert. Es wird ein Router-Objekt von Express verwendet, um spezifische HTTP-Methoden (GET, POST, PATCH, DELETE) mit entsprechenden Controller-Funktionen zu verknüpfen.



ABBILDUNG 10: BACKEND STRUKTUR

4.7.5 User Schema

Damit alle Benutzer einheitlich angelegt werden können, wird ein Mongoose Schema erstellt. Mit diesem Schema lassen sich die value Felder und dessen Konventionen Definieren. Beispielsweise kann definiert werden welch Datentyp ein Feld hat und welche Voraussetzungen erfüllt sein müssen, damit ein Eintrag gemacht werden kann. Dies ermöglicht, dass jedem neuen Benutzer

```
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, "Bitte geben Sie Ihren Vornamen an"],
  },
  email: {
    type: String,
    required: [true, "Bitte geben Sie Ihren Nachnamen an"],
  },
  username: {
    type: String,
    unique: true,
    required: [true, "Bitte geben Sie einen Nutzernamen an"],
  },
  password: {
    type: String,
    required: [true, "Bitte geben Sie ein Passwort an"],
    minlength: 8,
    select: false,
  },
  preferredAvatar: {
    type: String,
  },
  iban: {
    type: String,
    required: [true, "Bitte geben Sie eine IBAN an"],
    validate: [validator.isIBAN, "Bitte geben Sie eine gültige IBAN ein"],
  },
  balance: {
    type: Number,
    default: 1000,
  },
});
```

ABBILDUNG 11: USER SCHEMA

automatisch 1000 Geldeinheiten zur Verfügung stehen.

4.7.6 Controllers

4.7.6.1 AuthController

Hier werden wesentliche Authentifizierungs- und Autorisierungsfunktionen implementiert. Zuerst wird eine Funktion namens `signToken` definiert, die zur Erstellung eines JSON Web Tokens (JWT) dient, welcher die Identität eines Benutzers sichert und eine festgelegte Gültigkeitsdauer hat.

```
const signToken = (id) => {  
  return jwt.sign({ id }, process.env.JWT_SECRET, {  
    expiresIn: process.env.JWT_EXPIRES_IN,  
  });  
};
```

ABBILDUNG 12: SIGNIN TOKEN

In der `SignUp`-Funktion können sich neue Benutzer registrieren, indem ihre in der Datenbank gespeichert werden. Nach erfolgreicher Registrierung wird ein JWT erstellt und zusammen mit den Benutzerdaten zurückgesendet, was den erfolgreichen Abschluss des Vorgangs bestätigt.

```
exports.signup = async (req, res, next) => {  
  try {  
    const newUser = await User.create({  
      name: req.body.name,  
      email: req.body.email,  
      username: req.body.username,  
      password: req.body.password,  
      passwordConfirm: req.body.passwordConfirm,  
      preferredAvatar: req.body.preferredAvatar,  
      iban: req.body.iban,  
      balance: req.body.balance, // value of 1000 will be assigned in Model  
    });  
  
    const token = signToken(newUser._id);
```

ABBILDUNG 13: SIGNUP FUNKTION

Die login-Funktion ermöglicht es Benutzern, sich einzuloggen, indem sie ihre E-Mail und ihr Passwort überprüft. Nach der Verifizierung dieser Daten wird ebenfalls ein JWT generiert und ausgegeben, welcher für nachfolgende Anfragen zur Authentifizierung verwendet wird.

```
exports.login = async (req, res, next) => {
  try {
    const { email, password } = req.body;

    // 1) Check if email and password exist
    if (!email || !password) {
      return next(
        new appError("Bitte geben Sie E-Mail und Passwort ein.", 400),
      );
    }

    // 2) Check if user exists and password is correct
    const user = await User.findOne({ email }).select("+password");

    if (!user || !(await user.correctPassword(password, user.password))) {
      return next(new appError("Falsche E-Mail oder Passwort.", 401));
    }

    // 3) If everything is ok, send token to client
    const token = signToken(user._id);
```

ABBILDUNG 14: LOGIN FUNCTION

Die protect-Funktion dient als Middleware, die den Zugang zu bestimmten Routen einschränkt. Sie überprüft das Vorhandensein und die Gültigkeit eines JWT im Anfrage-Header, validiert diesen und stellt sicher, dass der zugehörige Benutzer noch existiert und sein Passwort seit der Token-Ausstellung nicht geändert hat. Bei erfolgreicher Überprüfung wird der Benutzer für die nachfolgenden Prozesse autorisiert, andernfalls wird der Zugriff verwehrt.

```
exports.protect = async (req, res, next) => {
  try {
    // 1) Getting token and check if it's there
    let token;
    if (
      req.headers.authorization &&
      req.headers.authorization.startsWith("Bearer")
    ) {
      token = req.headers.authorization.split(" ")[1];
    }

    if (!token) {
      return next(
        new AppError(
          "You are not logged in. Please log in to get access.",
          401,
        ),
      );
    }

    // 2) Verificaiton token
    const decoded = await promisify.jwt.verify)(token, process.env.JWT_SECRET);

    // 3) Check if user still exists
    const currentUser = await User.findById(decoded.id);
    if (!currentUser)
      return next(
        new AppError("The user belonging to this token does no longer exist"),
        401,
      );
  }
}
```

ABBILDUNG 15: PROTECTION FUNCTION

4.7.6.2 UserController

Hier werden verschiedene Funktionen zur Verwaltung von Benutzerdaten bereitgestellt. Die Funktionen interagieren mit einem Mongoose-Modell namens User und bieten eine Reihe von Endpunkten zur Bearbeitung von Benutzerinformationen.

Die Funktion `getAllUsers` holt alle Benutzer aus der Datenbank und sendet sie als Antwort zurück. Im Erfolgsfall wird ein Statuscode 200 zurückgegeben, zusammen mit einer Liste aller Benutzer und der Gesamtanzahl. Bei einem Fehler gibt die Funktion einen Statuscode 404 zurück und informiert den Client über das Problem.

```
exports.getAllUsers = async (req, res, next) => {
  try {
    const users = await User.find();

    // SEND RESPONSE
    res.status(200).json({
      status: "success",
      results: users.length,
      data: {
        users,
      },
    });
  } catch (err) {
    res.status(404).json({
      status: "fail",
      message: err,
    });
  }
};
```

ABBILDUNG 16: GETALLUSER ENDPOINT

Die `getUser` Funktion sucht einen spezifischen Benutzer anhand seiner ID. Wenn der Benutzer gefunden wird, wird er zusammen mit einem Erfolgsstatus zurückgegeben. Wenn kein Benutzer gefunden wird, sendet die Funktion eine 404-Antwort mit einer entsprechenden Fehlermeldung.

```
exports.getUser = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) {
      return res.status(404).json({
        status: "fail",
        message: "Kein Benutzer mit dieser ID gefunden",
      });
    }
  }
}
```

ABBILDUNG 17: GETUSER ENDPOINT

«updateUserBalance» ermöglicht es, das Guthaben eines Benutzers zu aktualisieren. Die Funktion prüft zunächst, ob die übergebene Balance-Änderung eine gültige Zahl ist. Anschliessend wird versucht, den Benutzer in der Datenbank zu finden und dessen Guthaben zu aktualisieren. Bei einem Fehler oder wenn der Benutzer nicht gefunden wird, wird eine entsprechende Fehlermeldung zurückgegeben.

```
exports.updateUserBalance = async (req, res, next) => {
  try {
    const userId = req.user.id;
    const { balanceChange } = req.body;

    if (isNaN(balanceChange)) {
      return res
        .status(400)
        .json({ status: "fail", message: "Must be a valid number" });
    }

    const user = await User.findById(userId);
    if (!user) {
      return res
        .status(404)
        .json({ status: "fail", message: "User not found" });
    }

    user.balance += balanceChange;
    await user.save();
  }
}
```

ABBILDUNG 18: UPDATEUSERBALANCE ENDPOINT

Die Funktionen «createUser», «updateUser» und «deleteUser» sind Platzhalter, die angeben, dass diese Routen noch definiert werden müssen. Sie geben derzeit einen Statuscode 500 zurück und informieren den Nutzer, dass die entsprechende Funktionalität noch nicht implementiert ist.

4.7.6.3 ErrorController

Dies ist ein Fehlerbehandlungsmodul, das verschiedene Arten von Datenbank- und Authentifizierungsfehlern abfängt und entsprechende Benutzerfehlermeldungen generiert.

4.8 Frontend

Angular ist ein modernes, umfangreiches Framework für die Entwicklung von Single-Page-Applications (SPAs), das von Google entwickelt und gewartet wird. Es ermöglicht Entwicklern, leistungsstarke und dynamische Webanwendungen durch die Verwendung von TypeScript, einer streng typisierten Version von JavaScript, zu erstellen. Angular nutzt ein Komponenten-basiertes Architekturmodell, das die Wiederverwendung von Code fördert und die Wartung erleichtert.

Hier werden die wichtigsten Elemente aufgelistet, die in dem Projekt verwendet werden.

TABELLE 59: ANGULAR ELEMENTE

Element	Beschreibung
Module	Diese bündeln verwandte Codebestandteile zusammen. Ein Modul definiert einen Rahmen, in dem bestimmte Komponenten zusammenarbeiten können
Komponenten	Diese sind die Bausteine der Anwendung und kapseln die Logik der Benutzeroberfläche, die Darstellung und das Verhalten in einer Klasse.
Services	Dienste sind wiederverwendbare Klassen, die spezifische Funktionen ausführen, wie z.B. Datenabruf von einem Server, und können in Komponenten injiziert werden, um deren Modularität und Wiederverwendbarkeit zu erhöhen.
Routing	Das Angular-Router-Modul ermöglicht die Navigation zwischen verschiedenen Ansichten und Komponenten innerhalb einer SPA, ohne die Seite neu laden zu müssen.

4.8.1 App.component.html

Beinhaltet HTML Elemente die auf allen Seiten angezeigt werden. In diesem Projekt wird die Navbar auf jeder Seite angezeigt. Darum wird sie hier implementiert. Zudem beinhaltet diese Seite den Router-Outlet. In Angular, das für die Entwicklung von Single-Page-Applications (SPAs) genutzt wird, wird jede Unterseite als sogenanntes "Outlet" im Router dargestellt. Der Router ist ebenfalls in dieser Struktur enthalten und ermöglicht die dynamische Anzeige verschiedener Ansichten innerhalb der gleichen Seite.

```
<navbar></navbar> <router-outlet></router-outlet>
```

ABBILDUNG 19: APPCOMPONENTHTML

4.8.2 App Routes

In dieser Datei werden die Routen definiert, die dem Router angeben, was er rendern soll. Dabei wurde für jede Page die geplant wurde eine route erstellt und dem routes-Array hinzugefügt. Dieses hat den tag «export» um in anderen Dateien referenziert werden zu können. Der erste Eintrag im Array ist ein leerer String. Dies bewirkt, dass wenn die Url leer ist, automatisch auf die «welcome» page weitergeleitet wird.

```
export const routes: Routes = [  
  { path: '', redirectTo: '/welcome', pathMatch: 'full' },  
  { path: 'welcome', component: WelcomeComponent },  
  { path: 'home', component: HomeComponent },  
  { path: 'login', component: LoginComponent },  
  { path: 'signup', component: SignupComponent },  
  { path: 'blackjack', component: BlackjackComponent },  
  { path: 'slots', component: SlotsComponent },  
];
```

ABBILDUNG 20: ROUTES ARRAY

4.8.3 Components

Dann wurden alle Components, die in diesem Fall die einzelnen Pages darstellen, erstellt und implementiert. Diese wurden im erstellten «components» folder angelegt, um den Code übersichtlich zu halten.

4.8.3.1 Home Component

Dieser dient als Hauptscreen nach dem man sich erfolgreich registriert und eingeloggt hat. Von hier aus kann man die einzelnen Spiele starten. Zudem werden hier Infos zum angemeldeten User gegeben.

4.8.3.2 Welcome Component

Dieser Screen wird neuen oder nicht angemeldeten Nutzern angezeigt. Dieser beinhaltet simplen Text um die Plattform anzupreisen.

4.8.3.3 Navbar Component

Hier werden kann man sich zum Homescreen navigieren oder sich registrieren bzw. an-/abmelden.

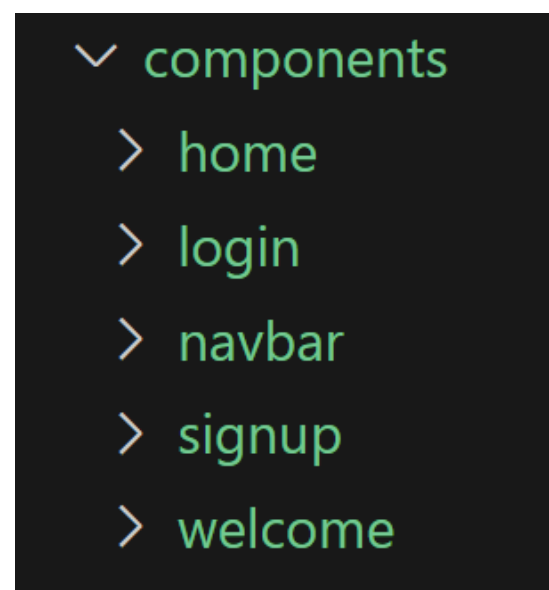


ABBILDUNG 21: COMPONENTS STRUKTUR

4.8.3.4 SignUp Component

```
signupForm = this.fb.group({
  name: ['', [Validators.required]],
  email: ['', [Validators.required]],
  username: ['', [Validators.required]],
  password: ['', [Validators.required, Validators.minLength(8)]],
  preferredAvatar: [''],
  iban: [
    '',
    [Validators.required, Validators.pattern(/DE\d{2}\s?(\d{4}\s?){4}\d{2}/)],
  ], // simple structure of german IBAN
});
```

In diesem Component wird die Registrierung abgewickelt. Dabei werden die Daten im Formular «signupForm», welches eine Verwendung der FormBuilder-Klasse ist, validiert und gespeichert.

ABBILDUNG 22: SIGNUPFORM

Dieses wird dann «onSubmit» an den Auth Service geschickt. Dort wird es dann an den API-Endpoint (/signup) geschickt, wo es dann vom Server validiert wird. Wenn dies erfolgreich geschehen ist,

```
signup(userData: any): Observable<any> {
  return this.http.post(`${this.baseUrl}/signup`, userData);
}
```

ABBILDUNG 23: SIGNUP AUTHSERVICE

schickt der Server die Daten an die MongoDB Datenbank.

4.8.3.5 Login Component

Dieser Component beinhaltet die Login Funktionalität und bearbeitet die Daten ähnlich dem SignUp Component. Die vom Benutzer eingegebenen Daten werden gebündelt. Dann werden sie über den AuthService an den (/login) API-Endpoint gesendet. Nach serverseitiger Validierung werden die Daten mit denen der Datenbank eingetragenen verglichen und bei Übereinstimmung authentifiziert.

4.8.4 Game Logik

Nachdem die Login- und Registrierungslogik im Backend wie im Frontend implementiert war, wurden die Spiele Blackjack und Slots erstellt.

4.8.4.1 Slot Maschine

Zuerst wurden Variablen initialisiert die den Gameloop ermöglichen.

```
reels: string[][] = [
  ['🍒', '🍋', '🍊', '🍉', '🍇'],
  ['🍒', '🍋', '🍊', '🍉', '🍇'],
  ['🍒', '🍋', '🍊', '🍉', '🍇'],
];

currentSymbols: string[] = [];
balance: number = 0;
bet: number = 5;
possibleBets: number[] = [5, 10, 20, 50, 100];
```

ABBILDUNG 24: SLOTMASCHINEN VARIABLEN

Beim Start des Programmes wird dann mithilfe des AuthService der Benutzerkontostand in das UserDetails interface übernommen. Der Kontostand ist nicht nur zum Abwickeln der Gewinne und Verluste wichtig, sondern auch um zu überprüfen, dass Spieler nicht ins Minus gehen können.

```
ngOnInit(): void {
  this.authService.fetchUserDetails().subscribe({
    next: (userDetails: UserDetails | null) => {
      if (userDetails) {
        this.balance = userDetails.data.user.balance;
      }
    },
  });
}
```

ABBILDUNG 27: SLOTMASCHINEN INITIALISIERUNG

```
interface UserDetails {
  data: {
    user: {
      id: string;
      balance: number;
    };
  };
}
```

ABBILDUNG 26: SLOTMASCHINEN USER INTERFACE

```
setBet(amount: number): void {
  if (amount <= this.balance) {
    this.bet = amount;
  }
}
```

ABBILDUNG 25: : SLOTMASCHINEN WETTÜBERPRÜFUNG

Beim Spinnen der Maschine wird in der «spin» Methode zuerst geprüft, ob der Wettbetrag den Kontostand des Nutzers überschreitet. Wenn dies nicht der Fall ist wird der Boolean «isSpinning» auf true gesetzt und die Drehanimation wird abgespielt. Um auf das Ende der Animation zu warten befindet sich der restliche Code innerhalb einer setTimeout Methode. In dieser wird in einem String Array durch alle Fruchtsymbole iteriert um drei verschiedene Symbole zu erhalten. Diese werden in «currentSymbols» zwischengespeichert um sie in der «calculateWin» Methode weiterbearbeiten zu können.

```
isSpinning = false;

spin(): void {
  if (this.bet > this.balance) {
    this.snackBar.open('Nicht genug Guthaben!', 'OK', { duration: 3000 });
    return;
  }

  this.isSpinning = true;

  setTimeout(() => {
    this.currentSymbols = this.reels.map((reel) => {
      const randomIndex = Math.floor(Math.random() * reel.length);
      return reel[randomIndex];
    });

    let winAmount = this.calculateWin();
    this.updateUserBalanceAfterWin(winAmount - this.bet);
  }, 1000);
}
```

ABBILDUNG 28: SLOTMASCHINE SPIN METHODE

Anschliessend wird in der «calculateWin» Methode der Gewinn bzw. Verlust berechnet. Dies geschieht indem man ein neues «uniqueSymbols» Set initialisiert und den «currentSymbols» Array übergibt. Mit einer einfachen If/ Else Abfrage wird nun überprüft, wie gross das Set nun ist. Wenn es nur einen Eintrag hat, heisst das, dass alle Symbole identisch sind und der Wettbetrag verfünffacht wird. Wenn es zwei Einträge gibt heisst das, dass zwei gleiche und ein anderes Symbol vorhanden sind. In diesem Fall, wird der Wettbetrag anderthalbfach gerechnet. Wenn kein Betrag gewonnen wurde wird ein Gewinnbetrag von 0 zurückgeschickt. Und man verliert seinen Wetteinsatz.

```
calculateWin(): number {
  const uniqueSymbols = new Set(this.currentSymbols);
  let winAmount = 0;

  if (uniqueSymbols.size === 1) {
    winAmount = this.bet * 5;
    this.snackBar.open('Grosser Gewinn!', 'OK', { duration: 3000 });
  } else if (uniqueSymbols.size === 2) {
    winAmount = this.bet * 0.5;
    this.snackBar.open('Kleiner Gewinn!', 'OK', { duration: 3000 });
  }

  return winAmount;
}
```

ABBILDUNG 29: SLOTMASCHINE CALCULATEWIN METHODE

Zuletzt wird der Verlust- bzw Gewinnbetrag direkt mit der Datenbank synchronisiert. Dafür kommt wieder Der «AuthService» verwendet.

```
updateUserBalanceAfterWin(balanceChange: number): void {
  this.authService.updateUserBalance(balanceChange).subscribe({
    next: () => {
      this.fetchLatestBalance();
    },
  });
}
```

ABBILDUNG 30: SLOTMASCHINE NUTZERKONTO ANPASSEN

Im «AuthService» erfolgt zunächst eine Überprüfung, ob «balanceChange» eine gültige Zahl¹ ist. Anschließend wird mittels der Benutzer-ID, die aus dem Token extrahiert wird, die korrekte URL für die API-Anfrage zusammengesetzt. Ein «HTTP PATCH-Request»² wird daraufhin gesendet, um das Benutzer Guthaben zu aktualisieren. Diese Anfrage erwartet eine Antwort, die ein Balance-Objekt enthält, welches dann den lokalen Kontostand aktualisiert. Dieser Prozess wird durch den Einsatz der Tap-Methode³ innerhalb einer Pipe-Funktion⁴ realisiert, um den Datenfluss gezielt zu manipulieren.

```
updateUserBalance(balanceChange: number): Observable<number> {  
  if (isNaN(balanceChange)) {  
    throw new Error('balanceChange muss eine Zahl sein');  
  }  
  
  const userId = this.getUserIdFromToken() || '';  
  const updateBalanceUrl = `${this.baseUrl}/${userId}`;  
  
  return this.http  
    .patch<{ balance: number }>(  
      updateBalanceUrl,  
      { balanceChange },  
      {  
        headers: {  
          Authorization: `Bearer ${this.getJwtToken()}`,  
        },  
      },  
    )  
    .pipe(  
      tap({  
        next: (response) => {  
          this.balance = response.balance;  
        },  
      })  
    );  
}
```

ABBILDUNG 31: AUTHSERVICE NUTZER KONTOSTAND AKTUALISIEREN

¹ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/isNaN?retiredLocale=de

² <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH>

³ <https://rxjs.dev/api/operators/tap>

⁴ <https://rxjs.dev/api/index/function/pipe>

4.8.4.2 Black Jack⁵

Anders als bei der Slot Maschine liess sich hier im HTML schon viel Logik mit «Angular Directives⁶» unterbringen.

Mit der `*ngFor`-Direktive⁷ und Dateninterpolation `{{ card.code }}`, können HTML-Elemente flexibel erzeugt werden. Jedes Array-Element wird in ein Bild konvertiert, das eine Karte darstellt, was eine visuelle Darstellung der Dealer-Karten ermöglicht.

```
<div class="dealer-hand">
  <!-- Zeige Dealer-Karten hier -->
  <div *ngFor="let card of dealerHand">
    
  </div>
</div>
```

ABBILDUNG 32: BLACKJACK DEALER STRUKTUR

Für die Benutzeroberfläche für den Spieler wurde ebenfalls Elemente erstellt. Hier können Spieler verschiedene Aktionen wie "Hit", "Stand", "Double Down" und "Split", abhängig vom aktuellen Spielzustand ausführen.

```
<div class="actions" *ngIf="!isGameOver">
  <button (click)="hit()" *ngIf="!playerDone">Hit</button>
  <button (click)="stand()" *ngIf="!playerDone">Stand</button>
  <button (click)="doubleDown()" *ngIf="playerHand.length === 2">
    Double Down
  </button>
  <button (click)="split()" *ngIf="canSplit()">Split</button>
</div>
<div class="game-over" *ngIf="isGameOver">
  <button (click)="startNewGame()">Neues Spiel</button>
</div>
```

ABBILDUNG 33: BLACKJACK AKTIONEN STRUKTUR

⁵ <https://www.spielbanken-bayern.de/spielinfos/spielregeln/black-jack#:~:text=Der%20Dealer%20muss%20bei%20einem,in%20der%20Höhe%20seines%20Einsatzes.>

⁶ <https://angular.io/guide/built-in-directives>

⁷ <https://angular.io/api/common/NgFor>

Diese Struktur ermöglicht es Spielern, Jetons aus einem Auswahlbereich zu nehmen und sie in einer Zone abzulegen, indem «Drag-and-Drop⁸» verwendet wird. Die Logik hinter den Event-Handle­rn und «removeJeton» wird im «JetonService», um die entsprechenden Daten zu verwalten und das UI entsprechend zu aktualisieren.

```
<!-- Jeton-Auswahlbereich -->
<div class="jetons">
  <img
    *ngFor="let value of jetonValues"
    class="jeton"
    [src]="assets/jetons/' + value + '.png'"
    [attr.data-value]="value"
    draggable="true"
    (dragstart)="onDragStart($event)"
  />
</div>

<!-- Drop-Zone für Jetons -->
<div
  class="jeton-drop-zone"
  (dragover)="onDragOver($event)"
  (drop)="onDrop($event)"
>
  Setzen Sie Ihre Jetons hier
  <div *ngFor="let jetonValue of setJetons; let i = index" class="set-jeton">
    
  </div>
</div>
```

ABBILDUNG 34: BLACKJACK JETONS STRUKTUR

Zuerst wurden Variablen initialisiert um die Logik implementieren zu können. Zudem wird das Nutzerguthaben wie bei der Slot Maschine gefetcht und in ein Interface übertragen.

```
public playerHand: Card[] = [];
public dealerHand: Card[] = [];
public isGameOver: boolean = false;
public playerDone: boolean = false;
public jetonValues = [5, 10, 20, 50, 100];
public balance = 0;
```

ABBILDUNG 35: BLACKJACK VARIABLEN

⁸ https://www.w3schools.com/jsref/event_ondragstart.asp

Dann wurde die die Spielinitialisierung programmiert Dabei werden zwei Karten an den Spieler verteilt und eine an den Dealer. Diese werden dann dem Array des Spielers und dem des Dealers hinzugefügt.

```
startNewGame(): void {
  this.gameService.newGame();
  const card1 = this.gameService.dealCard();
  const card2 = this.gameService.dealCard();
  const dealerCard = this.gameService.dealCard();

  if (card1 && card2 && dealerCard) {
    this.playerHand = [card1, card2];
    this.dealerHand = [dealerCard];
    this.isGameOver = false;
    this.playerDone = false;
  } else {
    this.snackBar.open('Fehler beim Starten eines neuen Spiels', 'OK', {
      duration: 3000,
    });
  }
}
```

ABBILDUNG 36: BLACKJACK SPIELINITIALISIERUNG

Wenn vom Spieler eine neue Karte gezogen wird, wird die «hit» Methode aufgerufen. Diese wiederum ruft im «GameService» die Methode «dealCard» auf. Wenn dies geschehen ist, wird überprüft ob der Spieler mehr als 21 hat oder nicht.

```
hit(): void {
  if (!this.isGameOver && !this.playerDone) {
    const card = this.gameService.dealCard();
    if (card) {
      this.playerHand.push(card);
      if (this.getHandValue(this.playerHand) > 21) {
        this.snackBar.open('Busted!', 'OK', { duration: 3000 });
        this.isGameOver = true;
      }
    } else {
      this.snackBar.open('Keine Karten mehr im Deck', 'OK', {
        duration: 3000,
      });
    }
  }
}
```

ABBILDUNG 37: BLACKJACK HIT METHODE

Im «GameService» wird mit der «pop⁹» Methode die letzte Karte des «deck»-Arrays zugewiesen. Die zugewiesene Karte wird im gleichen Zug aus dem Array gelöscht.

```
public dealCard(): Card {  
  const card = this.deck.pop();  
  if (!card) {  
    throw new Error('No more cards in the deck.');  }  
}
```

ABBILDUNG 38: GAMESERVICE DAELCARD METHODE

Danach wurde die Logik des Dealers implementiert. Dieser Zieht so viele Karten bis er auf 17 gekommen ist. Wenn eine Karte gezogen wurde wir sie dem «dealerHand»-Array hinzugefügt und der Gewinner wird überprüft.

```
dealerTurn() {  
  while (this.getHandValue(this.dealerHand) < 17) {  
    const card = this.gameService.dealCard();  
    if (card) {  
      this.dealerHand.push(card);  
    }  
  }  
  this.checkWinner();  
}
```

ABBILDUNG 39: BLACKJACK DEALERTURN METHODE

Zuerst wird der Gesamtwert der Hand berechnet, indem alle Kartenwerte summiert werden. Dann werden die Anzahl der Asse ermittelt, da Asse als eins oder elf gewertet werden können. Dann wird eine Schleife ausgeführt solange der Gesamtwert der Hand größer als 21 ist und es noch Asse gibt. Diese werden dann als eins gewertet.

```
private getHandValue(hand: Card[]): number {  
  let value = hand.reduce((acc, card) => acc + card.value, 0);  
  let aces = hand.filter((card) => card.value === 11).length;  
  
  while (value > 21 && aces > 0) {  
    value -= 10; // Ace can be 1 instead of 11  
    aces -= 1;  
  }  
  
  return value;  
}
```

ABBILDUNG 40: BLACKJACK GETHANDVALUE METHODE

⁹ https://www.w3schools.com/jsref/jsref_pop.asp

Die «checkWinner» Methode evaluiert den Ausgang einer jeden Runde, indem verschiedene Szenarien abgeglichen werden: Falls der Spieler einen Gesamtwert von über 21 erreicht, resultiert dies automatisch in einer Niederlage. Sollte der Dealer hingegen über 21 gehen oder der Spieler einen höheren Wert als der Dealer vorweisen, gewinnt der Spieler und erhält das Doppelte seines Einsatzes zurück. Verfügt der Spieler über einen niedrigeren Wert als der Dealer, verliert er ebenfalls seinen Einsatz. Endet das Spiel unentschieden, d.h., beide Parteien haben denselben Wert, bekommt der Spieler seinen Einsatz zurückerstattet.

```
private checkWinner(): void {
    const playerValue = this.getHandValue(this.playerHand);
    const dealerValue = this.getHandValue(this.dealerHand);
    this.isGameOver = true;

    let message = '';
    let balanceChange = 0;

    if (playerValue > 21) {
        message = 'Du hast verloren!';
    } else if (dealerValue > 21 || playerValue > dealerValue) {
        message = 'Du hast gewonnen!';
        balanceChange = this.jetonService.getTotalBet() * 2;
    } else if (playerValue < dealerValue) {
        message = 'Du hast verloren!';
    } else {
        message = 'Unentschieden!';
        balanceChange = this.jetonService.getTotalBet();
    }

    this.snackBar.open(message, 'OK', { duration: 3000 });
    if (balanceChange != 0) {
        this.updateUserBalance(balanceChange);
    }
}
```

ABBILDUNG 41: BLACKJACK CHECKWINNER METHODE

Die Karten sind ein Importiertes Asset (Siehe Kap, 1.4.3 Assets) und haben dabei die folgende Einteilung: Sie starten entweder mit einer Zahl von zwei bis zehn, oder den entsprechenden Kürzeln. Dabei steht «J» für Jack, «Q» für Queen, «K» für King und «A» für Ass. Gefolgt von einem Minus und der Kartenfarbe. «H» für Heart, «D» für Diamond, «S» für Spade und «C» für Clubs. Die Karten

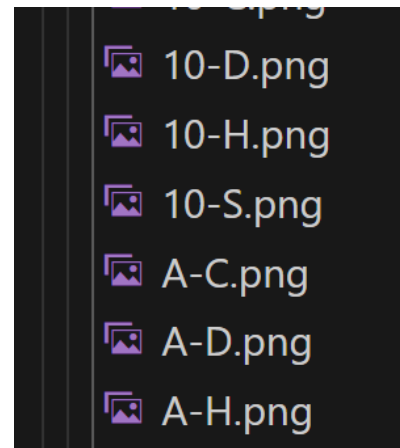


ABBILDUNG 42: KARTEN ASSETS

```
private getCardNumericValue(value: string): number {  
  if (value === 'A') {  
    return 11;  
  } else if (['J', 'Q', 'K'].includes(value)) {  
    return 10;  
  } else {  
    return parseInt(value);  
  }  
}
```

ABBILDUNG 43: BLACKJACK KARTEN EINBINDUNG

Schlussendlich wird der Gewinn bzw. Verlust wieder über den «AuthService» abgewickelt und per API an die Datenbank übermittelt.

```
private updateUserBalance(balanceChange: number): void {  
  this.authService.updateUserBalance(balanceChange).subscribe({  
    next: (newBalance) => {  
      this.balance = newBalance;  
      this.fetchBalance();  
    },  
  },
```

ABBILDUNG 44: BLACKJACK UPDATEBALANCE METHODE

5 Kontrolle

Die fünfte Phase ist der Kontrolle gewidmet.

5.1 Refactoring

Als alle geplanten Funktionen umgesetzt waren, wurde der Code nochmals auf Herz und Nieren per Whitebox¹⁰ Verfahren getestet. Überprüft wurde unter anderem, ob...

- ... die Code-Conventions, Formatierungen und Naming nach den Standards umgesetzt wurden
- ... der Code leicht verständlich und nicht verschachtelt ist
- ... die Konsole keine Irrelevanten oder Internen Ausgaben enthält
- ... das Frontend gut Strukturiert wurde

Nachfolgend sind die Hauptveränderungen aufgeführt.

5.1.1 Code Strukturierung

Einzelne Methoden in Komponenten wurden in Services aufgeteilt. So wurde aus der beispielsweise langen BlackJack Komponente eine kürzere Version mit Abhängigkeiten zu einem «blackjackService» und einem «jetonService».

5.1.2 Clean Code

Da die verwendeten Variablen und Methoden bereits gut benannt wurden, wurden diese nicht verändert. Jedoch wurden jegliche Funktionalitäten durch RxJS Operatoren verkürzt und deklarativer dargestellt¹¹. Dies verkürzte den Code erheblich.

5.2 Testdurchführung

Datum 05.05.2023

Person: Paul Pietzko

Für die Durchführung der Tests wurden wie geplant mit Jasmine und Karma durchgeführt. Dafür wurden einige Unit-Test programmiert.


¹⁰ <https://www.computerweekly.com/de/definition/White-Box-Test>

¹¹ <https://www.ionos.de/digitalguide/websites/web-entwicklung/deklarative-programmierung/>














































5.2.1 Basisarbeiten

TABELLE 60: TESTFÄLLE RESULTATE

Nr.	Zeitpunkt	Was wird getestet?	Erwartetes Ergebnis	Ergebnis
1.0	15:40	Registrierung eines neuen Benutzers mit fehlenden Angaben	Fehlermeldung mit Fehlendem Attribut.	<p>Die Felder zeigen eine Warnung an wenn sie falsch oder garnicht ausgefüllt wurden. Zudem lässt sich der Bestätigung Button erst betätigen wenn alle Felder korrekt ausgefüllt wurden.</p> <div> <div>IBAN*</div> <div>D75512108001245126199</div> <div>Bitte geben Sie eine gültige IBAN ein.</div> </div> <div> <div>Nutzername*</div> </div> <div> <div>Passwort*</div> </div>
2.0	15:47	Login Funktionalität mit falschen Angaben	Login wird verweigert und entsprechende Fehlermeldung wird übermittelt.	<p>Der Login findet nicht statt und man wird nicht weitergeleitet. Zudem wird dem Nutzer eine Snackbar mit einer entsprechenden Fehlermeldung angezeigt.</p> <div> <div>Login fehlgeschlagen. Bitte versuchen Sie es erneut. Schliessen</div> </div>
3.0	15:52	Aktualisierung des Kontostandes	Die Gewinne bzw. Verluste werden in Echtzeit mit der DB synchronisiert und angezeigt.	<p>Die Gewinne werden direkt mit dem Patch Api Endpoint an die DB geschickt und auch in der Applikation korrekt angezeigt.</p> <div> <div>PATCH /api/v1/users/65f2c7df75c40cd6a904f0f1 200 50.748 ms - 231</div> <div>Guthaben: 25000€</div> </div>

4.0	16:01	Spiellogik Überprüfung Blackjack	Alle Regeln werden korrekt umgesetzt und das Spiel ist komplett spielbar.	 <p>Das Spiel ist komplett spielbar. Die Gewinne und Verluste werden korrekt berechnet. Einsätze können korrekt getätigt werden. Jedoch fehlt die Split Funktion. Zudem werden die Karten vom Beginn an schon angezeigt, was im Normalfall nicht korrekt wäre, jedoch in diesem Spiel gewollt ist.</p>
6.0	16:07	Überprüfung Nutzerdaten im Backend.	Die Daten werden zusätzlich zur Clientsite Validierung auch im Backend validiert.	<p>Die Registrierung mit falschen Daten ist wie erwartet fehlgeschlagen. Die API gibt einen Status 400 Code zurück und das Frontend teilt dem Nutzer visuell mit, dass etwas fehlgeschlagen ist.</p> <pre>POST /api/v1/users/signup 400 355.333 ms - 136</pre> <p>Registrierung fehlgeschlagen. Bitte versuchen Sie es erneut. Schliessen</p>
7.0	16:14	Registrierung Funktionalität Frontend	Es ist möglich sich über die Website Oberfläche registrieren.	<p>Es ist möglich. Wie auch bei den anderen Aktionen wird auf der Serverkonsole wie auch im Frontend eine entsprechende Meldung gegeben</p> <pre>OPTIONS /api/v1/users/signup 204 0.194 ms - 0 POST /api/v1/users/signup 201 353.985 ms - 480</pre>

8.0	16:22	API SignUp End Point	Man kann über Postman mit korrekten Daten einen SignUp durchführen. Wenn der Versuch erfolgreich verläuft erhält man eine Success Message und einen JWT Token.	Der SignUp End Point funktioniert und gibt einen JWT Token wie auch eine Success Message zurück
9.0	16:30	Get User End Point.	Der Gesuchte Nutzer wird angezeigt.	Die Nutzer mit der entsprechenden ID wird korrekt angegeben.

10.0	16:42	GitHub Repository	Das Repository wurde korrekt angelegt und die Commits synchronisieren den Code und die Dateien korrekt auf dem Main Branch.	<div>Das GitHub Repository wurde korrekt angelegt und die Commits werden synchronisieren den Code wie gewollt mit dem Main Branch.</div> <table><tr><td></td><td>.vscode</td><td>initial commit</td><td>3 months ago</td></tr><tr><td></td><td>abgaben</td><td>05.05.2024</td><td>2 days ago</td></tr><tr><td></td><td>src</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>.editorconfig</td><td>initial commit</td><td>3 months ago</td></tr><tr><td></td><td>.env</td><td>Backend</td><td>2 months ago</td></tr><tr><td></td><td>.gitignore</td><td>initial commit</td><td>3 months ago</td></tr><tr><td></td><td>README.md</td><td>initial commit</td><td>3 months ago</td></tr><tr><td></td><td>angular.json</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>eslint.config.mjs</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>package-lock.json</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>package.json</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>server.ts</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>tsconfig.app.json</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>tsconfig.json</td><td>30.04.2024</td><td>last week</td></tr><tr><td></td><td>tsconfig.spec.json</td><td>30.04.2024</td><td>last week</td></tr></table>		.vscode	initial commit	3 months ago		abgaben	05.05.2024	2 days ago		src	30.04.2024	last week		.editorconfig	initial commit	3 months ago		.env	Backend	2 months ago		.gitignore	initial commit	3 months ago		README.md	initial commit	3 months ago		angular.json	30.04.2024	last week		eslint.config.mjs	30.04.2024	last week		package-lock.json	30.04.2024	last week		package.json	30.04.2024	last week		server.ts	30.04.2024	last week		tsconfig.app.json	30.04.2024	last week		tsconfig.json	30.04.2024	last week		tsconfig.spec.json	30.04.2024	last week
	.vscode	initial commit	3 months ago																																																													
	abgaben	05.05.2024	2 days ago																																																													
	src	30.04.2024	last week																																																													
	.editorconfig	initial commit	3 months ago																																																													
	.env	Backend	2 months ago																																																													
	.gitignore	initial commit	3 months ago																																																													
	README.md	initial commit	3 months ago																																																													
	angular.json	30.04.2024	last week																																																													
	eslint.config.mjs	30.04.2024	last week																																																													
	package-lock.json	30.04.2024	last week																																																													
	package.json	30.04.2024	last week																																																													
	server.ts	30.04.2024	last week																																																													
	tsconfig.app.json	30.04.2024	last week																																																													
	tsconfig.json	30.04.2024	last week																																																													
	tsconfig.spec.json	30.04.2024	last week																																																													

5.3 Testauswertung

Von den zehn Testergebnissen entsprachen zehn den Erwartungen. null Tests scheiterten. Bei einem gab es noch Verbesserungspotenzial.

Insgesamt kann man mit dem Ergebnis zufrieden sein. Da alle Funktionen Grundsätzlich implementiert wurden und funktionieren. Die Applikation funktioniert im gesamten. Beim BlackJack Spielmodus fehlt die «split» Funktion¹². Jedoch wurde für die Behebung dieses Mängel keine Zeit eingeplant, da er den Game loop nicht massgeblich stört und das Spiel ansonsten normal gespielt werden kann.

5.4 Verbesserungen

Nachfolgend wird beschrieben, wie die gefundenen Probleme Gelöst wurden.

Wie oben erwähnt fehlt die «split» Funktion. Jedoch ist diese in keinem Fall massgeblich um das Spiel spielen zu können. Da diese Funktion nur nebensächlich ist und die Zeit nicht reicht, wird sie nicht in das Spiel implementiert.

¹² <https://www.islandresortandcasino.com/blog/ask-the-dealer-what-does-double-down-mean-in-blackjack>

6 Auswertung

Die letzte Phase von IPERKA ist die Auswertung in der die Arbeit auf die Ausführung geprüft wird.

6.1 Vorgehen

6.1.1 Planung

Durch eine simple Planung mit auswertbaren Arbeitsschritten war die Überprüfung der Fertigstellung immer ein leichtes. So konnte das Projekt in sechs Meilensteine eingeteilt werden (siehe Kap. 2.1.5). Der erste Schritt bestand darin ein Angular Projekt anzulegen und die HTML Struktur der Screens zu erstellen. In einem weiteren war das Ziel, die API mit all ihren Endpunkten zu erstellen. Danach ging es darum die Gameloops der beiden Spielmodi zu programmieren. Die letzten Schritten waren das Refactoring der Applikation und das Abschliessen der Dokumentation

6.2 Probleme

Ein Projekt verläuft in den allerseltensten Fällen ohne Unannehmlichkeiten. Auch in diesem traten ein paar Probleme auf.

6.2.1 IPERKA

IPERKA war eine grosse Hilfe. Die Projektmanagementmethode brachte eine Struktur in das Projekt. Jedoch hat sie einen grossen Nachteil. Dadurch dass erst am Ende der Umsetzung getestet wird, kann nur erschwert auf gefundene Fehler eingegangen. In diesem Projekt verliefen alle Testfälle positiv jedoch konnte das Refactoring ebenfalls nur verkürzt durchgeführt werden, da die Zeit fehlte.

6.2 Zeitmanagement

Die Zeitschätzung war nicht sehr akkurat. Es wurde viel zu wenig Zeit für die Dokumentation eingeplant. Da dies ein relativ grosses Projekt ist, musste viel Zeit in dessen Umsetzung gesteckt werden. Diese fehlte dann bei der Dokumentation, die einen Grossen Teil der Bewertung ausmacht.

6.3 Einhaltung Meilensteine

Die zu Beginn definierten Meilensteine konnten bis auf einen nicht eingehalten werden. Die API hatte lange Zeit nicht funktioniert wie sie sollte, weshalb die anderen Meilensteine alle verrutscht sind.

TABELLE 61: MEILENSTEINE ÜBERPRÜFUNG

Meilenstein	Tag der geplanten Erreichung	Tag der Erreichung
Angular Anwendung Mit allen Screens	27.02	27.02
Rest API mit allen Endpoints	05.03	14.03
Blackjack Game Loop	12.03	14.03
Slots Maschinen Game Loop	14.03	26.03
Refactoring	26.03	16.04
Dokumentation	16.04	07.05

6.4 Resultat

Dads Endprodukt kann wie geplant von mehreren Benutzern gespielt werden. Dabei können sich neue Nutzer registrieren. Diese Daten werden dann in der Datenbank sicher gespeichert. Nachdem sich ein Kontostand eines Nutzers nach einer gespielten Runde verändert, verändert sich dieser auch in der Datenbank. Die Beiden Spielmodi sind vollkommen spielbar.

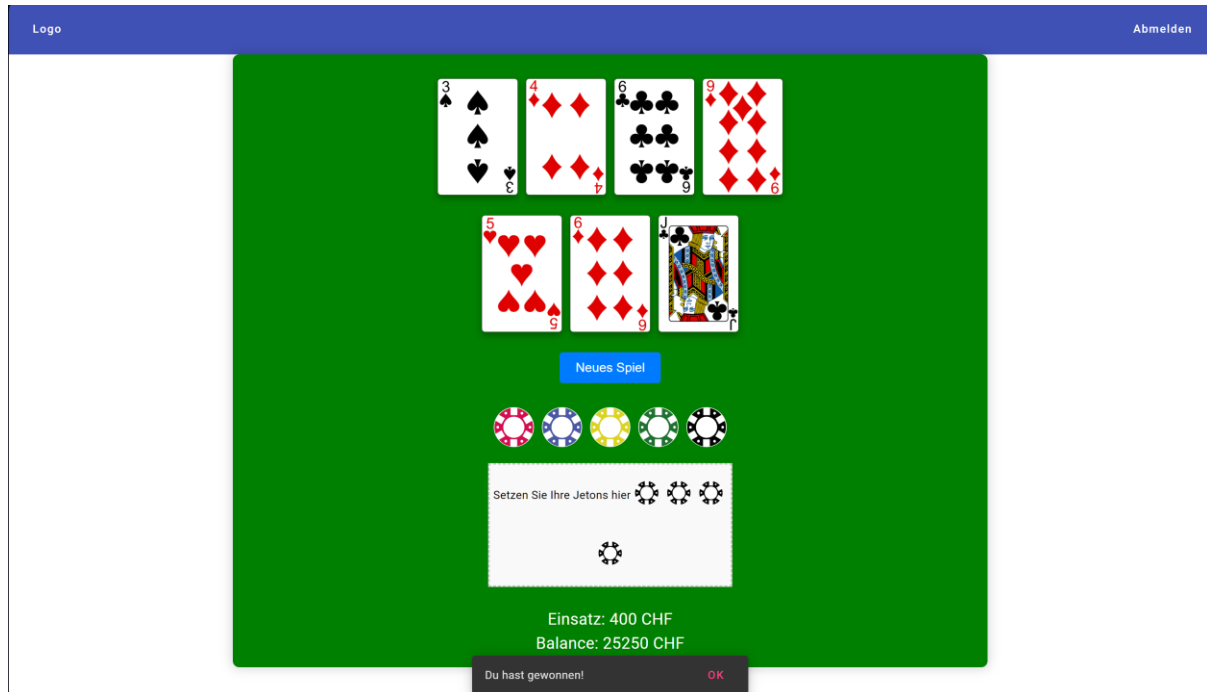


ABBILDUNG 45: RESULTAT BLACKJACK



ABBILDUNG 46: RESULTAT SLOT MASCHINE

6.5 Zukunft

In Zukunft könnte man noch mehr Spielmodi hinzufügen oder eine API erstellen die den Zugriff auf die Datenbank für andere Spielmodi Hersteller ermöglicht. So könnte man es diesen Entwicklern einfach machen ihre Spielmodi auf dieser Website zur Verfügung zu stellen.

Zudem könnte man die Applikation mit einem Echtgeldabwickler wie Stripe ausstatten. So könnte anstatt Falschgeld echtes Geld verwendet werden. So würde aus diesem Projekt eine richtige Applikation.

Im Code liesse sich noch einiges Verbessern. So könnte man die Logik in den einzelnen Komponenten auf einzelne Services noch weiter herunterbrechen. Die Jetons könnte man auch als Model darstellen um sie weiter abzuschirmen. Diese Änderungen würden den Code noch verständlicher und besser Strukturiert machen..

Schlusswort

Mein persönliches Ziel, alle geplanten Funktionen zu implementieren, konnte ich erreichen. Die App ist genau so benutzbar, wie ich es am Anfang definiert hatte. Zudem wollte ich in dieser Probe IPA eine möglichst realitätsnahe Dokumentation erstellen, was mir meiner Meinung nach auch gelungen ist.

Mit der Betreuung durch den EXP bin ich sehr zufrieden. Dieser half uns immer bei Fragen weiter und spielte sogar die EXP Gespräche durch. Zudem stellte er uns Schritt für Schritt die Standardkriterien vor. Dies half mir immer mich auf das Wichtige zu konzentrieren.

Während der Arbeit bestand selbstverständlich, aufgrund deren Wichtigkeit, etwas Druck. Dieser hat mich aber mehr angespornt als gestresst. Ich war ein Semester lang motiviert, um sowohl ein hervorragendes Produkt abzuliefern als auch eine Gute Note in der Dokumentation zu erzielen. Aus den Arbeitsjournalen ist zu entnehmen, dass ich zu Beginn etwas unsicher war was die IPA anging. Die ganzen Dokumente, welche zu erstellen waren, überforderten mich. Zudem haben wir noch nie eine Dokumentation in diesem Ausmass erstellen müssen.

Zudem sehe ich mich eher als Typ «Umsetzten». Es war schwer für mich alles im Voraus zu planen und Festzuhalten. Ich war sehr überfordert einzuschätzen wie lange mich ein Meilenstein kosten würde. Zudem war es schwer die gesamte Applikationsstruktur aufs Blatt zu bringen.

Rückblickend konnte ich so einiges von diesem Projekt lernen.

- Wie bereits beschrieben birgt die Arbeit mit IPERKA die Gefahr, dass Störgrößen erst gegen Ende des Projekts erkannt werden. In meinem Fall war dies jedoch nicht so schlimm, da ich jede Funktion vor ihrer Fertigstellung per Whitebox-Verfahren getestet hatte.
- Die Dokumentation hat einen viel höheren Stellenwert als ich gedacht hatte. Zu Beginn rechnete ich mit einem 50 zu 50 Verhältnis zwischen Dokumentation und Projekt. Eigentlich dachte ich das das zu viel Zeit für die Doku wäre. Jedoch war es eher ein 70 zu 30 Verhältnis. Viele Nachmittage gingen für das stumpfe tippen der Wörter drauf. Diese wären eigentlich für ein feineres Refactoring eingeplant gewesen.
- Oft ist es gut andere Meinungen einzuholen. Sei es in der Programmierung oder wenn es um das UI/UX im Frontend geht. Man selber weiss wo sich Dateien befinden und welche Methoden welche Funktion beherbergen. Für anderen sind die eigenen Gedankengänge jedoch nicht einsehbar, weshalb es viel Planung braucht um sein System, sei es von innen oder von aussen, für andere zugänglich zu machen.
- Während der Programmierung viel mir sehr auf, wie viel leichter es ist in einem Umfeld zu arbeiten, welches man schon kennt. Das Backend beispielsweise ist sehr umfangreich. Jedoch war es nicht allzu schwer zu vervollständigen, da ich dies bereits mehrere Male in der Vergangenheit bewältigt hatte.

Glossar

TABELLE 62: GLOSSAR

Begriff	Definition
API	Eine Schnittstelle, die es Anwendungen ermöglicht, untereinander zu kommunizieren.
Commit	Eine Aktion, bei welcher Änderungen an einem Code-Repository gespeichert werden.
Conventions	Übliche Standards und Regeln, die in der Softwareentwicklung befolgt werden, um eine konsistente und leicht verständliche Codebasis zu gewährleisten.
CSS	Cascading Style Sheets dient zur Gestaltung von Elementen auf Webseiten.
Destruktives Testen	Testmethode, die absichtlich Softwarefehler durch extreme oder ungewöhnliche Bedingungen provoziert, um Stabilität zu prüfen.
Fetchen	Eine Tätigkeit bei welcher Daten asynchron von einer API abgerufen werden.
Scrum	Agiles Rahmenwerk für iterative Entwicklungen von Softwareprodukten. Arbeit wird in festgelegten Zeiträumen sogenannten Sprints, durchgeführt.
Scrum	Scrum
SPAs (Single-Page Applications)	Webanwendungen, die auf einer einzigen HTML-Seite laden und dynamisch Inhalte aktualisieren. Benutzerinteraktionen erfordern kein vollständiges Seitenneuladen, was zu einer flüssigeren Benutzererfahrung führt.

Abbildungsverzeichnis

Abbildung 1: Zeitplan	7
Abbildung 2: Backend Struktur	25
Abbildung 3: API Endpoints.....	30
Abbildung 4: MVC Architektur	31
Abbildung 5: SLOt Maschine Architektur	32
Abbildung 6: BlackJack Game loop Architektur	32
Abbildung 7: Registrierung Architektur	32
Abbildung 8: Erstellung Repository.....	34
Abbildung 9: Prettier.....	35
Abbildung 10: Backend Struktur	36
Abbildung 11: User Schema	37
Abbildung 12: SignIn Token	38
Abbildung 13: SingUp Funktion.....	38
Abbildung 14: LogIn Function	39
Abbildung 15: Protection Function	40
Abbildung 16: GetAllUser EndPoint	41
Abbildung 17: GetUser EndPoint	42
Abbildung 18: UpdateUserBalance EndPoint	42
Abbildung 19: AppComponentHtml.....	43
Abbildung 20: Routes Array	44
Abbildung 21: Components Struktur	44
Abbildung 22: SignUpForm	45
Abbildung 23: SignUp AuthService	45
Abbildung 24: SlotMaschinen Variablen.....	46
Abbildung 27: SlotMaschinen Initialisierung	46
Abbildung 25: : SlotMaschinen Wettüberprüfung.....	46
Abbildung 26: SlotMaschinen User Interface	46
Abbildung 28: SLOtMaschine Spin Methode.....	47
Abbildung 29: SLOtMaschine CalculateWin Methode	48
Abbildung 30: SlotMaschine Nutzerkonto anpassen.....	48
Abbildung 31: AuthService Nutzer Kontostand aktualisieren	49
Abbildung 32: BlackJack Dealer Struktur	50
Abbildung 33: BlackJack Aktionen Struktur	50
Abbildung 34: BLACKJACK JETONS STRUKTUR.....	51
Abbildung 35: BlackJack Variablen.....	51
Abbildung 36: BlackJack SpielInitialisierung	52
Abbildung 37: BlackJack Hit Methode	52
Abbildung 38: GameService DaelCard Methode	53
Abbildung 39: BlackJack DealerTurn Methode	53
Abbildung 40: BlackJack GetHandValue Methode.....	53
Abbildung 41: BlackJack CheckWinner Methode	54
Abbildung 42: Kareten Assets	55
Abbildung 43: BlackJack Karten Einbindung	55
Abbildung 44: BlackJack UpdateBalance Methode	55
Abbildung 45: Resultat BlackJack.....	63
Abbildung 46: Resultat Slot Maschine	63

Quellenverzeichnis

MongoDB:

<https://www.mongodb.com/docs/>

JWT:

<https://jwt.io>

Angular:

<https://angular.io/docs>

RxJS Operators:

<https://rxjs.dev/guide/overview>

BlackJack:

<https://bicyclecards.com/how-to-play/blackjack/>